



Norwegian University of
Science and Technology

Towards adaptive coding tutorials

Emotion recognition in a programming
environment

Thor Håkon Bredeesen

Master of Science in Informatics

Submission date: June 2017

Supervisor: Asbjørn Thomassen, IDI

Norwegian University of Science and Technology
Department of Computer Science

Thor Håkon Bredesen

Towards adaptive coding tutorials

Emotion recognition in a programming environment

Master thesis, 2016/2017

Data and Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering



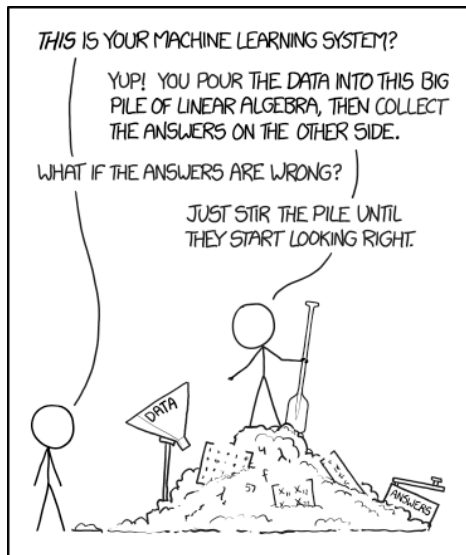


Figure 1: Comic by xkcd.com

Abstract

The research presented in this thesis aimed at non-intrusively detecting a learner's emotional state in a programming tutorial environment. These results could be used to adapt instructions and feedback to students in an online programming tutorial, potentially leading to more effective and motivating learning.

Detection of a learner's emotional state was done by collecting and analysing 23 participants' keystroke dynamics (how people type on their keyboard), and additionally pulse (heart rate) for five participants, in an online JavaScript tutorial developed for this research. Participants self-reported their emotional state, selecting one of six predefined states, hypothesised to be relevant for a learning situation. These emotions were: Bored, concentrated, confused, delighted, frustrated and surprised.

Both multiclass and binary classifiers were trained and tested on the dataset. In the binary classifiers, five classes were aggregated and classified against the sixth. Classification was tested on the whole population, and on individual participants. Every experiment was done with and without pulse features included to see if pulse influenced the classification.

Binary classifiers, using the whole population as a dataset, yielded the most promising results with accuracies ranging between 60% and 100%. Pulse was not found to give a better classification in this research. No conclusive results may be given however, as there are limitations in both the dataset and how the pulse feature was implemented. Still, this research does show promising results for non-intrusive emotion detecting in a programming environment.

Preface

This report is the result of one year's work on my master's thesis at the Department of Computer Science (IDI), at the Norwegian University of Science and Technology (NTNU). Asbjørn Thomassen has been the supervisor.

I would like to extend my thanks to the student association Online, for whom without I would have had more time to focus on the thesis, but less fun. Thanks also to: Michail Giannakos who threw me out on the deep end letting me give assignment lectures in IT2805 Web technologies; Marianne Kleveland who has been my closest supporter, motivating me and telling me to get my act together from time to time; Kathrine Steffensen and Therese Hansen Federl who proof-read all the exercises; All those who helped me solve programming challenges, especially Erik Frøseth and Sindre Johansen; To all my friends who let me talk about my thesis again and again, and all those who listened and challenged my ideas; Everyone who helped me by doing the tutorial, especially Ingrid Volden who recruited a few extra people, and those that attended the supervised data collection sessions; And lastly thanks to Asbjørn Thomassen for motivating me, sharing my ideas, dreaming big, letting me challenge myself, and laughing a bit.

Thor Håkon Bredesen
Trondheim, June 7, 2017

Contents

List of Figures	v
List of Tables	ix
Abbreviations	xi
1 Introduction	1
1.1 Background and Motivation	2
1.2 Goals and Research Questions	3
1.3 Research Method	4
1.4 Contributions	5
1.5 Thesis Structure	5
2 Background and motivation	7
2.1 Background theory	7
2.1.1 Emotions	7
2.1.2 Intelligent Tutoring Systems	11
2.1.3 Keystroke dynamics	12
2.1.4 Classification algorithms	13
2.1.5 Experiment metrics	17
2.2 Motivation	17
3 Related work	19
3.1 Emotion recognition	19
3.2 Keystroke dynamics	20
3.2.1 Keystroke dynamics as biometrics	20
3.2.2 Keystroke dynamics to recognise emotional states	22
3.2.3 Inferring a programmers performance and experience	28
3.3 Summary	28

4	Adapt: Keystroke dynamics collection tool	31
4.1	User interface	31
4.1.1	Collecting user information	33
4.1.2	Exercise handling	35
4.1.3	Inducing emotions	36
4.1.4	Interaction with the editor	37
4.2	System architecture	38
4.2.1	Key logging	39
4.2.2	Users	40
4.2.3	Exercises	42
4.2.4	Backend API	44
5	Data collection and experiments	45
5.1	Data collection plan	45
5.1.1	Participants	46
5.1.2	Data collection	46
5.2	Data collection setup	48
5.2.1	Supervised data collection setup	49
5.2.2	Personal information handling	50
5.3	Experiments	50
5.3.1	Nature of the dataset	50
5.3.2	Data classification tool	51
5.3.3	Experiment plan	57
5.4	Population overview	59
6	Results	61
6.1	Universal classifications	61
6.1.1	Universal multiclass classifier	62
6.1.2	Universal multiclass classifier with pulse	63
6.1.3	Universal subset multiclass classifier	64
6.1.4	Universal subset multiclass classifier with pulse	65
6.1.5	Universal binary classifier	66
6.1.6	Universal binary classifier with pulse	68
6.2	Individual classifications	70
6.2.1	Individual multiclass classifier	72
6.2.2	Individual binary classifier	73
7	Evaluation, contributions and future work	79
7.1	Evaluation and discussion	79
7.1.1	Data collection phase	80
7.1.2	Experiment phase	83
7.1.3	Research question 1	86

7.1.4	Research question 2	90
7.1.5	Is it possible to determine a learner’s emotional state non-intrusively in a programming tutorial?	91
7.2	Contributions	91
7.3	Future work	92
Appendices		95
A	Exercises	97
B	Application to Norwegian Centre for Research Data	119
C	Participant contract	125
D	Response from Norwegian Centre for Research Data	129
E	Aggregated feature vectors by class	133
Bibliography		161

List of Figures

1	Comic by xkcd.com	2
2.1	Two-dimensional valence-arousal space, adapted from Russell [2003].	8
2.2	Mapping emotional states to the two-dimensional space by Russell [2003], adapted from Baker et al. [2010]	9
2.3	Learning cycle in a two-dimensional space, adapted from Kort et al. [2001].	10
2.4	Unknown instance classified as a blue square using KNN, with $K = 4$	14
2.5	Unknown instance classified as a red circle using SVM.	15
2.6	By adding a new feature, it is possible to find a linear decision boundary in a higher dimensional space.	16
4.1	Adapt's initial view	32
4.2	Adapt's user information registration form	33
4.3	Adapt's emotion registration form	34
4.4	Error message display when the result is wrong	35
4.5	Error message when the code contain errors	36
4.6	The timers three stages.	37
4.7	Feedback form for emotion inducing features.	38
4.8	Keystroke capture flow	39
5.1	Screenshot from the supervised data collection session.	49
5.2	Relationship between key press time and key flight time for two keystrokes, adapted from Shukla and Solanki [2013].	55
6.1	Comparison of all KNN classifier combinations for the universal multiclass classifier.	64
6.2	Comparison of all SVM classifier combinations for the universal multiclass classifier.	65

7.1	Comparison of average feature vectors for all six classes.	84
7.2	Average feature vector for bored, with the minimum and maximum values	85
7.3	Average feature vector for concentrated, with the minimum and maximum values	86
7.4	Average feature vector for confused, with the minimum and maximum values	87
7.5	Average feature vector for delighted, with the minimum and maximum values	88
7.6	Average feature vector for frustrated, with the minimum and maximum values	89
7.7	Average feature vector for surprised, with the minimum and maximum values	90

List of Tables

3.1	The feature set used by Epp [2010].	24
3.2	The feature set used by Kołakowska [2015].	26
4.1	Adapt’s session model with example	41
4.2	Adapt’s user model with example	42
4.3	Adapt’s exercise model	43
4.4	Adapt’s API endpoints	44
5.1	Pattern used to recognise sequence of characters	51
5.2	Features used in this thesis. Note that the pulse features are only used when they have been collected.	54
5.3	Information about the participants, order by sessions	60
6.1	Class distribution	62
6.2	Results from the universal multiclass classifier	63
6.3	Results from the universal multiclass classifier using pulse	66
6.4	Results from the universal subset multiclass classifier	67
6.5	Results from the universal subset multiclass classifier with pulse	68
6.6	Results from the universal binary classifier	69
6.7	Results for the universal binary classifier with pulse	70
6.8	Individual class distribution	71
6.9	Information about individual participants	71
6.10	Individual multiclass classifier for participant A	73
6.11	Individual multiclass classifier for participant B	73
6.12	Individual multiclass classifier for participant C	74
6.13	Individual multiclass classifier for participant H	75
6.14	Individual binary classifier for participant A	75
6.15	Individual binary classifier for participant B	76
6.16	Individual binary classifier for participant C	76
6.17	Individual binary classifier for participant D	76

6.18	Individual binary classifier for participant F	76
6.19	Individual binary classifier for participant H	77
E.1	Mean, min and max values for aggregated bored feature vectors . .	133
E.2	Mean, min and max values for aggregated concentrated feature vectors	138
E.3	Mean, min and max values for aggregated confused feature vectors	142
E.4	Mean, min and max values for aggregated delighted feature vectors	146
E.5	Mean, min and max values for aggregated frustrated feature vectors	151
E.6	Mean, min and max values for aggregated surprised feature vectors	155

Abbreviations

API	=	Application Programming Interface
ATS	=	Affective tutoring systems
BACH	=	Brain Automated chorales
CBT	=	Computer-based training
CSS	=	Cascading Style Sheet
EEG	=	Electroencephalogram
HTML	=	HyperText Markup Language
ITS	=	Intelligent tutoring system
KNN	=	K-nearest neighbours
NSD	=	Norwegian Centre for Research Data
NTNU	=	Norwegian University of Science and Technology
PNN	=	Probabilistic Neural Network
SD	=	Standard deviation
SVM	=	Support vector machine
XML	=	Extensible Markup Language

Chapter 1

Introduction

Towards adaptive coding tutorials

In courses with a large amount of students, it would seem impossible to adapt the tutoring style to each individual student. All of these students do not only have different amounts of experience, they also have their own style of learning. Moreover there might be variances in motivation for each student from day to day.

If the courses could have been complemented with a tutorial application that adapted to student's physiological and learning state, and gave appropriate instructions and feedback for a given state, students could be motivated and learn more efficiently.

One way to indicate a person's emotional state is through their typing rhythm. Typing rhythm is how, and not what, they type on a keyboard. Previously it has been experimented with detecting a person's emotional state through typing rhythm in daily activities. However, in a learning situation, this approach has not been tested before.

This master's thesis aim to classify student's emotional state through the use of typing rhythm, and additionally pulse for a few participants. The work done in this study could lay the groundworks for moving towards an adaptive tutoring platform, to complement lectures.

In this chapter, an overview will be given first on relevant background theory and motivation behind this study. Then the goal and research questions will be posed. In the following section, the research method will be presented, followed by the contribution from this thesis. Lastly the thesis' structure will be presented.

1.1 Background and Motivation

As a complementary tool to regular lectures, online programming tutorials could be used to teach students to code on their spare time, and in their own pace. However, the online programming tutorials available today are constructed linearly, meaning that everyone are taught the same way regardless of previous experience and motivation. This could lead some students to give up if they don't understand the curriculum, or get bored if they aren't challenged.

Previously, It has been shown that it is somewhat possible to detect a person's current emotional state by analysing how they type on their keyboard (Epp [2010]). However, this has been tested on a number of emotional states not relevant during learning. Baker et al. [2010] summarises six emotional states that should be considered in a learning situation: Bored, concentrated, confused, delighted, frustrated and surprised. Detecting these emotional states would make it possible to adapt feedback to a student's needs, motivating and helping them in an appropriate manner.

Tutoring systems that adapt to a student's cognitive state, and tries to model their knowledge, are called intelligent tutoring systems. These systems make use of cognitive science, learning science, artificial intelligence and mathematics to create an environment able to adapt to a student's way of learning. Picard [1997] introduced the term affective tutoring systems, which are intelligent tutoring systems able to adapt their teaching style to a student's affective or emotional state.

In this research, this was sought to be achieved using non-intrusive methods, such as the computer's keyboard. By detecting a learner's keystroke dynamics, it would be possible to classify a sequences of characters as one of the six proposed emotional states. This classification was done using the two instance-based classification algorithms k-nearest neighbours and support vector machine.

These algorithms share some similarities, in that each data sample (instance) is located in a feature space (which can be any n-dimensional space). During a training phase, instances with known classifications are used to assign certain areas in the space to a class. When unknown instances are introduced in the space, the already known instances are used to classify the new instance.

For k-nearest neighbours this is done by having a number of the closest instances to the unknown instance vote on its classification. In support vector machine, known instances are used to create borders between different classifications in the space. Which side of the border the unknown instance is placed decides its classification.

The researcher's background is as a computer science student, for whom programming did not come easy. During his studies he has been involved with the development of exercises for the introductory course IT2805 Web technologies.

In this course, first year students learn HTML, CSS and JavaScript. During the later years, he has also contributed to collecting and writing the curriculum, and lecturing one hour each week in assignment lectures.

The work done by Yuksel et al. [2016] sparked the interest for this thesis. In their study they measured participant's cognitive workload to determine if the participant were ready to go to the next challenge when learning to play music composed by Bach.

Feedback from students taking IT2805 made it apparent that there are differences in how challenging exercises and curriculum are to different students. If the researcher could help students learn in a way adapted to them, it would be a great achievement. The approach taken by Yuksel et al. [2016] does not scale well for a course with 200 students, as each participant needs a dedicated device. Using the keyboard to detect emotional states however, as tested by Epp [2010] and Kołakowska [2015], inspired a non-intrusive method to achieve this goal.

1.2 Goals and Research Questions

The overarching goal for this thesis is:

Goal *Can a learner's emotional state be detected non-intrusively in a programming tutorial?*

This goal is a step toward creating an adaptive programming tutorial for first year students. In this thesis, the aim is to recognise a student's emotional state in a way that can be scaled to any course, regardless of size. Methods for emotion recognition that uses any specialised equipment, e.g. electroencephalography, would be too expensive to deploy in a course with more than just a few students. It would probably also make it impossible for students to do exercises when they want to. By finding a non-intrusive way to detect emotions, i.e. a method that students does not notice, would thus be preferable.

As a way to reach the goal stated above, two research questions were asked:

Research question 1 *Is it possible to detect a learner's emotional state using keystroke dynamics from programming keywords?*

The chosen method for this thesis, as it has shown promising results as a source of emotion recognition, is keystroke dynamics. By recording a person's keyboard usage, i.e. which keys are pressed, when they are pressed, and for how long, it could be possible to see differences between emotional states.

Even though there are a number of ways to solve a problem with coding, programming languages have a strict grammar which is important to know. Keywords are an important part of this grammar, and important to know in order to master the language.

Programming keywords are reserved words in programming languages that can only mean one thing. It can be compared with words in a written language. Some keywords are used quite often, and are thus a good data source for emotion recognition, since only similar strings of text can be classified against each other.

Research question 2 *Will detecting a learner’s pulse yield a better classification than typing rhythm alone?*

KM et al. [2015] found that pulse somewhat could be used to assess whether a person was in a positive or negative state. Bahreini et al. [2016a] combined different data sources to assess people’s emotional state, which yielded a far higher accuracy than those of the separate data sources alone. Therefore, this thesis aims to see if adding pulse as a data source could increase the accuracy of the classification.

This should also be done using a non-intrusive method, e.g. smartwatch, activity tracker, or using the integrated webcam to read changes in skin colour.

1.3 Research Method

This research focused on analysing the collected keystroke and pulse data, and experiment with different classification methods. Both k-nearest neighbours and support vector machine classifiers were tested with different hyperparameters, different sets and subsets of the collected data, and two different approaches: Multiclass and binary classifiers. This was done both to see if it was possible to classify the selected emotional states based on the data made available, and to find the best approach to do so within the limits of this thesis.

The data was collected using Adapt, a programming tutorial resembling Codecademy (www.codecademy.com), built for this thesis. Adapt registers three features for each keystroke that was made: the key’s unique identifier, the key’s press time, and the key’s release time. This data was later used to create a feature vector that could be used in the experiments.

In order to get as many participants as possible, it was possible to do the tutorial without observation, i.e. participants could do it on their own time, whenever they liked. However, some were invited to do the tutorial while being observed by the researcher in order to collect pulse data, and merge this data with the keystrokes.

There exists no openly available keystroke dynamics dataset used to classify emotional states that the researcher is aware of. Hence it was necessary to collect and create such a dataset for this research.

1.4 Contributions

The results and contributions to the research field will further be discussed in chapters 6 and 7.

Three of the selected emotional states have previously not been classified using keystroke dynamics. These were the emotions concentrated, confused and delighted. The highest classification score for these were 60%, 79% and 100% respectively, using a binary classifier. However the dataset used in these experiments were limited, as such the results themselves should not be seen as conclusive. What this research can conclude however, is that using keystroke dynamics as a basis for non-intrusive classification in a programming tutorial shows promise for emotions relevant to learning.

Additionally, the research found that using the support vector machine algorithm is more promising than using k-nearest neighbors, but other classification algorithms should be considered.

In this study, adding pulse features did not have any major effect on the classifications.

1.5 Thesis Structure

This thesis is structured as follows: In chapter 2, background theory on key subjects for this thesis are presented, and a motivation for why the researcher chose to pursue the research is given. Chapter 3 presents previous work and research that closely relates to this thesis. In chapter 4, the data collection tool Adapt is presented. Chapter 5 contains information on the data collection process and how the experiments were conducted. The results from the experiments are found in chapter 6. Lastly, in chapter 7, the research is evaluated and discussed, contributions to the research field is presented, and some ideas for future work are proposed.

Chapter 2

Background and motivation

In order to understand the research presented in this thesis, a number of key subjects must be defined and explained. These are presented in this chapter's first section. The subjects are emotions, intelligent tutoring systems, keystroke dynamics and classification algorithms. The second and last section gives a rationale for why the author chose to conduct this research.

2.1 Background theory

This section explains the key subjects emotions, intelligent tutoring systems, keystroke dynamics and classification algorithms, all relevant for this thesis.

2.1.1 Emotions

Emotions can traditionally be described in two ways, either as cognitive or behavioural changes (Cannon [1927]). The cognitive approach suggests that emotions are experienced in the brain, independent from bodily sensations. The behavioural approach, on the other hand, focuses on physiological responses as an expression of emotions. The responses can be pulse, blood pressure and respiration rate. However, Picard [1997] state that recent approaches see these two in combination, that emotions can be described both as cognitive activity and as physical changes in the body.

The terms mood and emotion are often used interchangeably, however, there is a significant distinction between them. Emotions are short lived and triggered by either a physical or cognitive cause, whereas mood is long lived, more subtle and either positive or negative (Picard [1997]). This thesis only concerns itself with emotions and how they are expressed. Participants in this study were asked

to register their current emotional state, as it was perceived by them. Emotional state is a person’s current emotion. This data from the users were used to label data samples.

2.1.1.1 Classifying emotions

In the literature review for this thesis, two main approaches for classifying emotions have been found: they can either be classified as discrete categories, or in a continuous two-dimensional space.

Discrete categories are based on how we define emotions through language, i.e. when talking about our emotions we give labels to specific emotional episodes. Such labels could be, but are not limited to, concentrated, nostalgic, satisfied, frustrated, angry, bored and confused. Perhaps most famous are the six basic and universal emotions presented by Ekman et al. [1978]: fear, anger, happiness, sadness, disgust and surprise. These emotions are hypothesised to be universally expressed across all cultures. Zimmermann et al. [2006] states that definitions of emotional classes do not only vary among languages, but they also vary within one language. This is a challenge with using discrete classes, as different people might define emotions differently, and therefore classify them differently.

An approach to counter this limitation is to describe emotions in a continuous two-dimensional space, as coordinates of valence and arousal (Lang [1995] and Russell [2003]), as seen in figure 2.1.

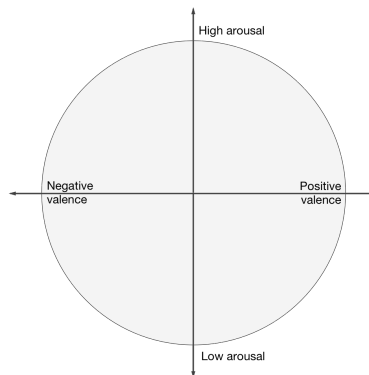


Figure 2.1: Two-dimensional valence-arousal space, adapted from Russell [2003].

Valence is the general description of an emotion, if it’s positive or negative;

or pleasant or unpleasant. Arousal is how active the emotion is, thus the axis is often referred to as activation, and it is either high or low.

2.1.1.2 Emotions in learning

Graesser et al. [2007] have doubted the relevance of the six basic emotions presented by Ekman et al. [1978] for learning, as it is unlikely that a student experience e.g. fear while in a learning situation. Emotional states that are hypothesised to influence cognition and learning are summarised by Baker et al. [2010]. These emotional states are: boredom, confusion, delight, engaged concentration, frustration and surprise.

Figure 2.2 visualise the mapping between the set of emotions summarised by Baker et al. [2010] and the two-dimensional valence-arousal space.

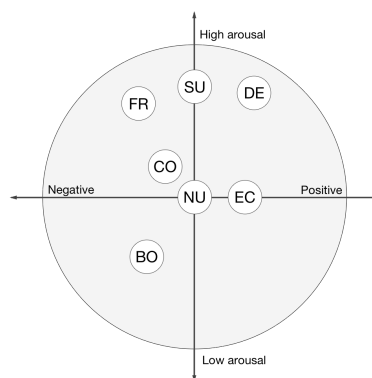


Figure 2.2: Mapping emotional states to the two-dimensional space by Russell [2003], adapted from Baker et al. [2010]

BO: boredom, EC: engaged concentration, CO: confused, DE: delighted, FR: frustration, SU: surprise, NU: neutral.

Engaged concentration, as mentioned in figure 2.2, is described by Csikszentmihalyi [1990] as “flow”, meaning that a person’s concentration is intense, that they are focused and deeply involved with the task at hand.

Kort et al. [2001] proposed a model to regard the complex interplay between emotions and learning. This model can be viewed in figure 2.3. It depicts a learning cycle consisting of two axes: learning and emotion. The learning axis ranges from un-learning (which is negative) on the bottom, to constructive learning at the top. The emotion axis goes from negative emotions (such as confusion

and disappointment) to positive emotions (such as satisfaction and hopefulness). Each quadrant describes a student's emotional state, and how a teacher should intervene to best help the student.

In quadrant I a student is happily engaged in exploratory learning. While the student is in this quadrant there is no need to intervene. In quadrant II the student is starting to encounter difficulties caused by misconceptions or an incomplete understanding of the subject. Subtle intervention from a teacher might be necessary, but only to fill in the gaps or errors in the student's mental model. In quadrant III the student has acknowledged that they had been working on the basis of an incomplete model. Intervention from a teacher at this stage should be supportive, reassuring the student that they will emerge from the disappointment. In the last quadrant (IV), the student is back to basic, constructing and improving their understanding of the subject. Intervention at this point is equal to quadrant II, with subtle hints to lead students on the correct path.

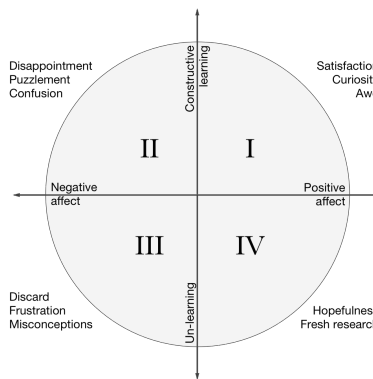


Figure 2.3: Learning cycle in a two-dimensional space, adapted from Kort et al. [2001].

The model states that students usually start in the the two top quadrants (I and II), and move in a counterclockwise direction. The research done by Kort et al. [2001] indicated that learning and emotions are not stable throughout the learning process. Despite the variance in emotions, however, it is consistent that students do experience different emotional states as they go from un-learning to constructive learning.

It was hypothesised that it is common to see a transition from confusion to frustration, and confusion to boredom. However, no empirical evidence for

this was given by Kort et al. [2001]. Perkins and Hill [1985] proposed another transition model, where frustration led to boredom. This hypothesis was based on data that showed these two states to be associated, but not that they were temporally related (that boredom precedes frustration).

The selected emotional classes used in this thesis are the ones summarised by Baker et al. [2010]: boredom, confusion, delight, engaged concentration, frustration and surprise. Engaged concentration was redefined in this thesis as concentrated in an effort to use words participants in the study would find familiar. This hopefully lead to a lesser degree of them having to interpret what was meant.

2.1.2 Intelligent Tutoring Systems

Although developing or testing an intelligent tutoring system (ITS) was not a part of the research conducted in this thesis, they are a part of the greater context.

An ITS is a digital system that adapts its teaching to the user in some fashion. They make use of computational models from sciences like cognitive science, learning science, artificial intelligence and mathematics to create a learning environment that can model, learn and improve from a learner's' psychological states, and give individual instructions. The interaction between the student and the system evolves dynamically over time, based on the learner's and the system's constraints (Graesser et al. [2012]). This means that an ITS can adapt to the student, and should teach in a more tailored way than e.g. a linear tutorial like Codecademy.

ITS can be divided into four approaches (Ma et al. [2014]):

- **Expectation and misconception tailoring:** Where learners communicate with the system using natural language and the system calculate misconceptions about a subject from the learner's answer.
- **Model tracing:** Here the system models the user's knowledge using ACT-R theory. A probable path through the knowledge set is calculated for the user, given their actions or answers. The learner's knowledge can thus be traced.
- **Constraint-based modeling:** Given a task and a constraint, the learner gives an answer or does an action. If the answer/action is outside of the constraint, the learner is given a feedback from the system.
- **Bayesian network modeling:** This uses probabilistic reasoning to calculate the probability that a learner has an understanding or misunderstanding of a concept.

Although there are different types of ITSs, the key definition is that they track the user’s knowledge and give adaptive instructions. This is contrary to conventional computer-based training (CBT), where some might adapt to individual learners, but does so following simple learning principles. There are, however, no sharp borders between ITS and CBT (Graesser et al. [2012]).

In 1997, the term affective tutoring systems (ATS) was introduced by Picard [1997]. ATS refers to an intelligent tutoring system that is able to tailor the teaching style to the user’s affective state (a person’s feeling or emotional state). E.g. if a user is bored, the system could present more challenging tasks, hopefully improving the student’s satisfaction and learning, and save time.

In the meta-analysis on ITS performed by Ma et al. [2014], it was shown that ITSs are associated with significantly higher achievement outcomes than any other learning method, except for small-group human tutoring and individual human tutoring. The author notes, however, that the ITSs analysed are evaluated relative to their scope. As such they should not replace other tutoring methods currently in use.

2.1.3 Keystroke dynamics

Keystroke dynamics is the process of measuring and assessing a person’s typing rhythm. In short, it is how, and not what, we type on a keyboard (Teh et al. [2013]). The typing rhythm is measured by observing a keyboard, which specific keys are used, and when they are pressed and released. Throughout this thesis, the terms typing rhythm and keystroke dynamics are used interchangeably.

During World War II, it was discovered that military units could be tracked, not only by visual confirmation, but also because it was possible to identify the unit’s telegraph operators by their keying rhythm. This was called “the fist of the sender” (Dunstone and Yager [2008]). Similarly the timing of keyboard events have been used to identify a computer user. This was first proposed in 1980 by Gaines et al. [1980]. This research found that typists appeared to have a signature while typing, and that this signature could be used as a basis for an authentication system.

Research on authentication systems using keystroke dynamics uses machine learning techniques to classify a sequence of keystrokes against a reference model (Monrose and Rubin [1997]). The benefit of using keystroke dynamics as an authentication tool is that the keyboard already is present, so you don’t need any additional hardware. In addition, as it is an already integrated part of the workflow, it is non-intrusive, meaning that users probably won’t alter their behaviour. Because of this, it is also possible to capture data in stealth mode, meaning that you can do authentication continuously (Mondal and Bours [2014]).

Monrose and Rubin [2000] hypothesised that users change their typing rhythm

according to their environment, stress level and cognitive state. This hypothesised connection between cognitive state and behaviour (e.g. typing rhythm) can be used to infer a person's emotional state (Kołakowska [2013]). This approach does not only concern itself with identifying one specific person's typing rhythm from the others', but identifying one specific person's different typing rhythms.

2.1.4 Classification algorithms

Kołakowska [2013] summarises a number of classification algorithms that have been tested to classify keystroke data, including C4.5 decision trees, artificial neural networks, Bayesian networks, AdaBoost, statistical analysis and k-nearest neighbours.

This research used the two classification algorithms k-nearest neighbours and support vector machines, the reason for which will be evident in the section 3.2.2. K-nearest neighbours and support vector machine will be presented in the two subsequent subsections.

2.1.4.1 K-nearest neighbours

K-nearest neighbour (KNN) is an instance-based classification algorithm (Mitchell [1997]), meaning that training instances (known data-class pairs) are situated in a feature space (a n-dimensional space). When a new unknown instance is added to the space with training instances, the K closest instances already present in the space vote on the unknown instance's classification. K is a constant set by the researcher. In its most simple form, the majority vote decides its classification. Each instance consists of a number of features, and these features are collectively called a feature vector, which looks like:

$$[f_0, f_1, f_2, (\dots), f_{n-1}, f_n]$$

In figure 2.4a, a 2-dimensional feature space with training instances is illustrated. An instance with an unknown classification is then introduced in the space, as seen in figure 2.4b. The K (in this case $K = 4$) nearest neighbours to the new instance vote on the unknown instance's classification. Since three out of four neighbours are blue squares, the unknown instance is classified as a blue square, as can be seen in figure 2.4c. Should, however, $K = 1$, only the closest neighbour gets to vote, and the unknown instance's classification will be equal to its nearest neighbour. In that case the unknown instance would be classified as a red circle.

It is possible, however, to alter the voting such that not all k nearest neighbours have the same influence, but those instances that are closer have a greater

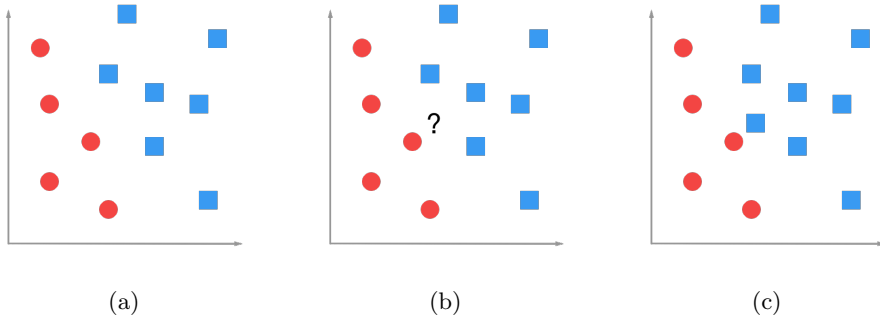


Figure 2.4: Unknown instance classified as a blue square using KNN, with $K = 4$

influence on the classification. This weighting is called distance weight, while the equal influence weight is called uniform weighting.

The weight is one of several hyperparameters, the value for K being another, and the distance metric is a third. These hyperparameters define the algorithm, and are used to tune the algorithm so that it does not become too specific nor too general for a certain scenario.

Euclidean distance is one of the most common distance metrics, Manhattan distance is another. Both metrics can be expressed through the Minkowski distance, where the value for p decides if it is the Euclidean or Manhattan distance. The value for p would be 2 and 1 respectively.

$$\left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

The KNN algorithm does not assume a general hypothesis about the data, unlike e.g. a decision tree where only a subset of features might be used to classify unknown instances. KNN, however, uses all features when calculating the distance between two instances. This is both a strength and weakness. In some data sets only a subset of the features might be relevant for a classification, in such cases KNN would calculate the distance using irrelevant features. On the other hand, KNN is robust against instances with incorrect values for certain features (noisy instances), given a large enough training set and feature vector. Noisy instances would then loose in the vote against other instances. It is possible to overcome the problem of irrelevant features by:

1. Weighting the features such that important features have a high weight, or
2. removing irrelevant features during preprocessing.

2.1.4.2 Support vector machine

Support vector machine (SVM) is also an instance-based classification algorithm (Cortes and Vapnik [1995]). This algorithm tries to find a linear separator line (a hyperplane for higher dimensional spaces) between instances of different classes. Not only does it find a separator, but it will find the one with maximum distance to the instances, yielding the largest margin possible between two classes. The instances used to create the separator are called support vectors. Data points that are not support vectors can mainly be disregarded during classification as they do not affect the separator line. A two-dimensional feature space can be seen in figure 2.5a. In figure 2.5b a linear separator is introduced, dividing the two classes. Then in figure 2.5c an unknown instance is added, and classified as a red circle because it is positioned to the right of the separator line. Had the separator line not have had the largest margin possible, and instead been tilted more to the right, the new instance could have been falsely classified as a blue square.

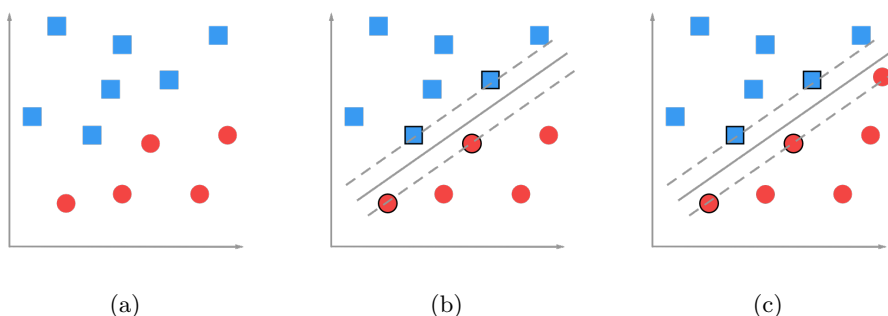


Figure 2.5: Unknown instance classified as a red circle using SVM. The solid line is the separator line, while the dotted lines are the margins.

The large margin allows for a safety zone, so that the classification algorithm has a higher tolerability for noisy instances.

In some cases, it might not be possible to find a linear separator. By applying the kernel trick, however, features can be transformed so that this is possible, in a higher dimension (Cortes and Vapnik [1995]). The kernel trick is a method to add new features to the data samples, derived from the features already present. This increases the number of dimensions, hopefully making it possible to find a linear separator in the higher dimensional space.

In figure 2.6a, a two-dimensional feature space has instances that are not linearly separable. By applying the kernel trick, this case adding the feature z (see figure 2.6b), the feature space can be transformed to a three-dimensional

space. In this higher dimensional space it is possible to apply a linear separator between the blue squares and the red circles, as seen in figure 2.6c.

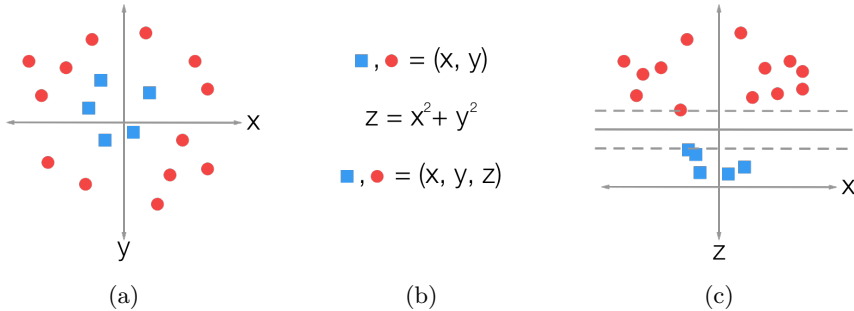


Figure 2.6: By adding a new feature, it is possible to find a linear decision boundary in a higher dimensional space.

The solid line is the separator line, while the dotted lines are the margins.

When applying the kernel trick, new features are not actually added to the vector, but the dot product (inner product) for the features used to create the new feature is calculated. This is done using a kernel function. Two notable kernels are Radial Basis Function (RBF) and the linear kernel.

The choice of kernel is one of several hyperparameters for SVM. Another hyperparameter is C , which is used to set how wide the decision boundary's margin should be. This is a tradeoff between having a smooth and simple decision boundary, and classifying all the training instances correctly. A small value for C allows constraints to be ignored, resulting in a larger margin and a more general classifier. On the other hand, a high value for C classifies more training instances correctly, but also yields a less general separator, which may lead to overfitting (a classifier that is too specific to the given training instances).

Kernel and C are two parameters that always must be considered. For some kernels other hyperparameters might be added. For RBF, there is a third hyperparameter called gamma. Gamma defines how far the influence of a single training instance reaches. With a low gamma value, training examples have a far reach, while the opposite is true for high gamma values. This means that training values can affect the decision boundary's margin. For RBF it is necessary to tune hyperparameters C and gamma based on the dataset.

2.1.4.3 KNN versus SVM

KNN is generally slower than SVM, because all instances in the feature space is considered when a classification is required. The poor performance in time

can be reduced by choosing an appropriate data structure, such as a KD-tree. With KNN, no calculations are done until an unknown instance is introduced, hence the training time is minimal. If the dataset consist of a large amounts of instances, and the vectors have few features, KNN is often preferred over SVM.

SVM, on the other hand, handles outliers well, and is more time efficient than KNN. By default, it can't categorise instances that aren't linearly separable in its current dimension. However, as explained, by applying the kernel trick, SVM can go around this limitation. If there are few instances in a high dimensional space, SVM is more preferred than KNN.

The dataset presented in this study doesn't clearly prefer one of the classification algorithms presented. Thus both are used in the experiments.

2.1.5 Experiment metrics

In the experiments presented in chapter 6, a number of metrics are used to described how well the classifiers performed. These metrics are precision, recall and Cohen's kappa coefficient (kappa score) (Witten et al. [2016]).

Precision Precision is the percentage of instances classified as a given class (true positives and false positives) that is truly that class (true positives). For a classifier that classifies six instances as bored, when only four of them were true positives (actually bored), the precision is 67%.

Recall Recall is the percentage of correctly classified instances out of all those instances that belong to a given class. For a classifier that classifies four instances of bored correctly (true positives), but there are seven instances of bored in the dataset (true positives and false negatives), the recall is 57%.

Kappa score Kappa score is a measure of how well a classifier classify instances versus chance alone. The score is given as a number between -1 and 1. If the score is 0, the classification is as good as guessing (random). With a score less than 0, it is actually better to guess than using the classifier. With a score of 1, there is a perfect agreement, i.e. every instance was correctly classified. Kappa score is only a measure for how well one classifier performed. In the study by Epp [2010], a kappa score greater than 0.4 was seen as acceptable.

2.2 Motivation

The paper that sparked the author's interest for this thesis was the study Brain Automated Chorales (BACH) presented by Yuksel et al. [2016]. In the BACH

study, researchers measured the participant's oxygenation level in the anterior prefrontal cortex to determine their cognitive workload. When the measured workload was below a given threshold, indicating that users had mastered the task at hand, the users were allowed to proceed to more challenging tasks. This way, BACH adapted to the learner's cognitive state. As stated by the authors, the premise of BACH can be applied to learning situations where tasks can be broken into increasing levels of difficulty.

From the summer of 2014, this thesis' author has taken part in creating, supervising and reviewing exercises in the course IT2805 Web technologies. IT2805 is a web development introductory course, i.e. students learn basic HTML, CSS and JavaScript. During the fall of 2016, the author also gave exercise lectures once a week. For some of the students the exercises were easy, and for others they were more challenging. Because of the amount of students taking part in the course (about 200), it was not possible to write exercises, nor give lectures, that universally fitted every student. That would have been too demanding on the course's resources.

However, if exercises and curriculum could be presented as an intelligent tutoring system, adapting to students cognitive workload, students could go ahead with the course in their own pace, and hopefully be more motivated and learn more efficient. Rather than having experienced students become bored, and those with no experience feeling overwhelmed.

The approach presented by Yuksel et al. [2016] would be difficult and expensive to scale up to 200 students. However, a non-intrusive method that would detect students' workload could scale better. In the search for such a method of measurement, the author came across a paper by Leinonen et al. [2016]. They found that it is partially possible to distinguish between novice and experienced programmers using keystroke dynamics. However, as the researcher has experienced himself, not only programming experience affect how well you learn, but also mood, emotional state and environment has an effect.

Detecting a learner's emotional state could thus be of help to adapt instructions and feedback, and motivate students to continue learning (Sarrafazadeh et al. [2008]), possibly also for those having a bad day. This is certainly something the researcher would have had great use of, as learning to code did not come easy.

The researcher thus wanted to find a way to detect learner's cognitive or emotional state using non-intrusive methods. These findings could later be used to develop a tool for courses like IT2805, giving large masses of students a learning environment adapted to them.

Chapter 3

Related work

This thesis draws experience from, and builds upon, previous work on emotion recognition and keystroke dynamics. Relevant previous research is presented in this chapter. In the first section, some research on emotion recognition is presented. The second part of this chapter addresses research on keystroke dynamics. In the third part, a short summary of the important research is given, and the choice of classification algorithms is discussed.

3.1 Emotion recognition

Below, three researches on emotion recognition are presented. The first one uses an intrusive approach, but it indicates that emotion recognition indeed is possible. The second and third research uses a non-intrusive approach to detect computer user's emotional state.

In the study by Kao et al. [2015] Electroencephalogram (EEG) was used to detect eight types of positive and negative emotions: Joyfulness, angeriness, protected, sadness, surprised, fear, satisfaction and unconcerned. The study found that all the negative emotions had a greater energy than the positive, and that it is possible to detect all selected emotions, except for surprise and fear which shared the same brainwave characteristics.

KM et al. [2015] used a multimodal approach to detect user's emotional state by combining keystroke dynamics, written text semantics and variations in the pulse. To detect the pulse they used a video feed of the user to extract the skin's color variations, indicating the user's pulse. During the testing they did not actually combine these three sources, but tested them separately. Keystroke dynamics and text semantics were tested on five and nine discrete emotional classes respectively, while pulse was used to determine if the user's emotional

state was positive or negative. The experiments resulted in an average accuracy of 77.67%, 88.70% and 73% for keystroke dynamics, text semantics and pulse respectively.

Bahreini et al. [2016a] needed real-time emotion recognition in their online communication skills training tool. To achieve this, they used facial recognition through the webcam, and analysed how users spoke through their microphone. Participants were asked to mimic the six basic and universal emotions stated by Ekman et al. [1978]. From each source, a feature vector was created and classified. The results from both classifiers were combined and classified again. This research used two expert raters to individually evaluate and label the data. The multi-modal approach resulted in an accuracy of 98.6%. Previously the researchers had tested unimodal emotion recognition on facial recognition (Bahreini et al. [2016b]) and voice recognition (Bahreini et al. [2016c]) alone, which yielded an accuracy of 72% and 67% respectively.

3.2 Keystroke dynamics

The research presented in this section has been divided into two categories: Keystroke dynamics as biometrics to infer users identity, and Recognise user's emotional state based on keystroke dynamics. These two categories are not mutually exclusive, as there are overlapping approaches and results.

Both user detection and emotion recognition requires unknown data to be classified. When using keystroke dynamics to authenticate users, the task is to classify keystrokes from one person against all other people. In emotion recognition, however, keystrokes need to be classified against keystrokes from the same person. Unless there exists some universal attributes for each emotional state, reflected in how we type, that is.

3.2.1 Keystroke dynamics as biometrics

User authentication is the most dominant use case for keystroke dynamics. There are three types of authentication (Teh et al. [2013]): knowledge based (something a person knows, e.g. a password), token based (an object in a person's possession, e.g. access card) and biometric based (a person's physiological or behavioral characteristics, e.g. fingerprint or voice). Keystroke dynamics falls within the latter category and is a behavioral characteristics.

This approach is already in use. Both the education website Coursera and the Norwegian bank DNB uses keystroke dynamics to authenticate users. Coursera uses it for selected courses where they offer users identification (<https://www.coursera.org/about/privacy>), and DNB uses it to authenticate users

after they have logged into their online bank account (Bjørndal and Bakken [2015]).

Monrose and Rubin [2000] built three classifiers to automatically authenticate users using Euclidean distance measure, non-weighted probability, and weighted probability. From a dataset of 63 users the accuracy ranged from 83.22% to 92.14%. The researchers argued that even though behavioral traits are a sign of identity, it has some limitations. E.g. a user's typing rhythm is a result of the user and the environment (e.g. different keyboards and emotional states change how we type). But when keystroke dynamics is implemented together with other ways of identification, e.g. a knowledge based approach, it allows for a more robust authentication system.

Revett et al. [2007] applied methods of machine learning to classify whether a user was authentic or not. The study had 50 participants, 20 acting as authentic users and 30 acting as imposters. The authentic users were told to write a chosen login id and password three times a day during a 14-day period. The imposters were told to log into each of the 20 authentic accounts four times each. The researchers then compared their modified Probabilistic Neural Network (PNN) to the approach by Sung and Cho [2006] who used SVM (which resulted in a 8-10% error rate). Using the modified PNN resulted in an error rate of 4%, compared to 8% for the standard PNN algorithm. Compared to a multilayered feedforward neural network, the modified PNN was superior in both training time and accuracy. It was noted, however, that PNN's efficiency would degrade as the number of samples grew, and that in this case the numbers were low. On the other hand, other methods such as backpropagation, required a large amount of training data, which PNN did not. Revett et al. [2007] stated that one of the most critical issues was to extract the correct parameters when using keystroke dynamics as a biometric. In their study, no feature selection was done, but they found that derived attributes, e.g. digraph and trigraph times, had an higher information gain than attributes such as flight time (the time between one key is released and the next is pressed) and dwell time (the time between one key is pressed and released).

Mondal and Bours [2014] explored continuous authentication, i.e. the user's behavior was monitored while logged in, and the user's authenticity was calculated using fuzzy logic methods. They recorded keystroke dynamics and mouse actions for 52 participants over 5-7 days in an uncontrolled environment. All users were genuine users. They applied different classifiers on different features, Scaled Euclidian Distance was used on single keys, digraphs, single and double mouse clicks. Correlation distance was used on digraphs and double mouse clicks. On mouse movement artificial neural network and counter-propagation artificial neural network was used as classification methods. The researchers implemented a trust model which compares the user's behaviour with a template of the gen-

uine user. The trust was continuously calculated, and given predefined fuzzy rules, the user would or would not be locked out. The results from this research were promising, but there was need for further studies. The main challenge was the membership functions, as these should be optimised for one user, or even for different actions.

Longi et al. [2015] used keystroke dynamics to identify students in a programming course. As far as the researchers knew, that was the first time user recognition was used in the context of programming. The goal was to recognise users within the same course, but also in a later course, as to see if the keystroke dynamics would change over time. Students participating in the study could use different computers and keyboards during the data collection, which have been found to have an effect on identification accuracy (Villani et al. [2006]). The researchers used a KNN classifier, with an Euclidean distance measure. Data samples were collected during two consecutive courses, each lasting 7 weeks. The calculated features from the data samples were divided into four levels: level 0 (average dwell time of any key), level 1 (average time the programmer needed to reach a specific key on the keyboard), level 2 (average time the programmer needed to press a specific key-pair, i.e. a digraph) and level 3 being a combination of the three mentioned.

To identify students within the same course, data from weeks 1 through 6 was used as a training set, and data from week 7 was used as a test set. During weeks 1 through 6, 233 students participated, while during the 7th week 173 students participated. The best results were found for the level 2 features, with accuracy from 90.8% ($K = 1$) and 95.4% ($K = 5$) to 97.7% ($K = 10$).

To identify students between courses, all events from the first course were used as a training set, and all events from the second course were used as the test set. Using $K = 1$ and level 2 features, they got an result of 98.6%. With $K = 2$ they only failed to identify one out of 146 students. As stated by the researchers, one reason for the high score might be the large amount of data available. To detect how student's typing change over time, the researchers used the first two weeks of the first course as a training set, and the two last weeks of the second course as a test set. There were eleven weeks between the training and test set. 70.8% were correctly identified when $K = 1$ and 90.8% were correctly identified when $k = 10$.

3.2.2 Keystroke dynamics to recognise emotional states

In the subsequent sections, previous researches that closely relate to this thesis will be presented in detail. The two first article reviews, by Epp [2010] and Kołakowska [2013], are divided into four subsections: data collection, data representation or feature selection, classification and results. Other relevant researches

on keystroke dynamics will be presented after those two.

Kořakowska [2013] summarises previous work on keystroke dynamics and mouse movement, and how this can be used in emotion recognition, citing eleven articles. These articles present results that ranges from 62% to 94% in accuracy. Three of the articles are set in an intelligent tutoring systems context. One of them only partially used keystroke dynamics, while the other two used mouse movements as a data source. Therefore, these three articles are not referenced in this thesis.

3.2.2.1 Emotion recognition by Epp [2010]

The research done by Epp [2010] is a master’s thesis written at the University of Saskatchewan, Canada. The thesis has later been summarised in the paper by Epp et al. [2011].

Data collection For the thesis, a data collection application was developed and ran in the background on users computers, gathering keystroke events regardless of which application was currently in use. At different times during the day, the application prompted the user with the keystrokes from the last 10 minutes, an emotional state questionnaire, and with an fixed text to type. Users could choose to opt-out of the data collection at any time, e.g. in case they were busy or had typed sensitive information. The questionnaire contained 15 5-point Likert scale questions regarding the user’s current emotional state. Users were asked to rate each emotion from “strongly disagree” to “strongly agree” on how it represented their current emotional state. The emotional states used were: frustrated, focused, angry, happy, overwhelmed, confident, hesitant, stressed, relaxed, excited, distracted, bored, sad, nervous and tired. The fixed text was a random piece from Alice’s Adventures in Wonderland (Carroll [1898]). The random piece of text was not selectable, so that users couldn’t copy and paste the text.

26 people initially participated in the study, but not all of them completed enough samples to be included in the analysis. The study thus ended with 12 participants. The software that collected the keystroke data was installed on the user’s private computers, and gathered data during their daily activities.

Feature selection From the provided data, three categories of information were extracted: keystroke features, emotional state classes and additional data points. The keystroke features consisted of key press and release events, unique codes for each key and a timestamp for when the key event occurred. The keystroke features were derived from the timing of single keystrokes, digraphs (two-letter combinations) and trigraphs (three-letter combinations). The article does not give any information on how many characters each piece of text consisted

of, but the feature vector grew to over 100.000 features, leading the researcher to only use aggregated features. The features used by Epp [2010] are shown in table 3.1.

Applied to	Description
Digraphs	The duration between the 1st and 2nd down keys of the digraph
	The duration of the 1st key of the digraphs
	Duration between 1st key up and next key down of the digraphs
	The duration of the 2nd key of the digraphs
	The duration of the digraphs from 1st key down to last key up
	The number of key events that were part of the graph
Trigraphs	The duration between 1st and 2nd down keys of the trigraphs
	The duration of the 1st key of the trigraphs.
	Duration between 1st key up and next key down of trigraphs
	The duration between 2nd and 3rd down keys of the trigraphs
	The duration of the 2nd key of the trigraphs
	Duration between 2nd key up and next key down of trigraphs
	The duration of the third key of the trigraphs.
	The duration of the trigraphs from 1st key down to last key up.
The number of key events that were part of the graph	

Table 3.1: The feature set used by Epp [2010].

The keystroke features used were the keystroke duration (dwell time) for each key, digraph and trigraph; keystroke latency (flight time) for each key, digraph and trigraph; and keystroke features that combine aspects of the duration and latency features. Additionally the number of mistakes (backspace and delete key) were used as a feature.

Each data sample was labeled by the user using the 5-point Likert scale and 15 discrete emotional classes. The additional data points are the active process name for each collected keystroke, so that the research can distinguish between different applications, and analyse the data in the context of the application.

Classification Because of the large variations in number of responses per user, no user-specific model was created. Instead data was aggregated across participants into one dataset. The classification method used was the supervised learning algorithm C4.5 decision tree. The previously described features were evaluated and selected using the correlation-based feature subset attribute selection method by Hall [1999], as not all features had the same information gain in the classification. After excluding data samples from users with too few responses (less than 50), the researchers were left with 1129 valid data samples to analyse. As this was a limited data set, 10-fold cross-validation was used (Wit-

ten et al. [2016]). Responses were not distributed evenly among emotion classes, thus under-sampling was utilised, removing instances from majority classes which resulted in more uniform class distribution.

Results The research found that a binary classifier was the best approach for the classes confidence, hesitance, nervousness, relaxation, sadness, excitement, anger and tiredness, with accuracies ranging between 77.4% and 87.8%.

The thesis notes that the dataset’s limitations in size and class distribution should be taken into account.

3.2.2.2 Emotion recognition by Kołakowska [2015]

The research by Kołakowska [2015] focuses mostly on individual classifiers, experimenting with different classification algorithms, and a smaller set of emotion classes than the one presented in Epp [2010].

Data collection Data collection was accomplished using an application that ran in the background on the Windows operation system. This application recorded all keyboard events. After a specified number of keyboard events had occurred, the user was presented with a questionnaire containing seven radio buttons, one for each predefined emotional states: happiness, sadness, boredom, anger, disgust, surprise and fear. In addition there was a textbox, where users could enter any other emotion. After selecting an emotional state, the keystroke data, and the result from the questionnaire, were saved to an XML file. The application needed to be started again after each questionnaire so that user’s voluntarily shared their keystroke data, and wasn’t disturbed by the questionnaire.

Kołakowska gathered 9 participants, and the data was gathered while the they were doing their daily activities. The researcher notes that data could be manipulated by users, as self-reporting was used to label the data. Self-reporting, however, was important for the participants, as it gave them the possibility to decide when the keystrokes were recorded and not. The data collection was based on the researcher’s “trust in the participants’ motivation and goodwill”.

Feature selection Only free text was collected from the users, thus a number of words to compare were needed. From a frequency dictionary for the Polish language, 20 two-character words and 20 three-character words were found, making it possible to create digraphs and trigraphs from the data collected. The digraphs and trigraphs are complete Polish words, not parts of words, as was the case with Epp [2010]. The features extracted are similar to Epp [2010]: dwell time, flight time, duration of key sequences and the time between subsequent

keys. In addition, the means and standard deviation were calculated for all features, and frequencies of the keys “backspace”, “delete”, “enter” and “spacebar” were calculated. The features are shown in table 3.2.

Applied to	Description
Single keys	Dwell time
Digraphs	Dwell time for the first key
	Dwell time for the second key
	Time between pressing the first and the second key in a digraph
	Flight time between the first and the second key
	Digraph duration (time between pressing the first and releasing the second key)
	Number of events for a digraph (usually 4)
Trigraphs	Dwell time for the first key
	Dwell time for the second key
	Dwell time for the third key
	Time between pressing the first and the second key in a trigraph
	Time between pressing the second and the third key in a trigraph
	Flight time between the first and the second key
	Flight time between the second and the third key
	Trigraph duration (time between pressing the first and releasing the third key)
Number of events for a trigraph (usually 6)	

Table 3.2: The feature set used by Kotakowska [2015].

Classification No unique universal subset of features that could be used to discriminate between emotions were found in the dataset. However, using a feature filtering criterion, some features were found to discriminate between emotions for one user. This subset of features were different for other users, however. Thus concluding that there were no global features distinguishing between emotions in this dataset. All 36 original features were used during the classification.

Decision trees, neural networks, KNN, naive Bayes, AdaBoost, rotation forest and Bayesian networks were all trained and tested. Similar to Epp [2010] under-sampling a k-fold cross-validation (where k was a value between 20 and 30 depending on the number of classes for a given user) was used as the dataset was unevenly distributed among classes and small in size. This research tried to build both a universal and an individual identifier.

Results Results for the universal binary classifier ranged from 47.37% to 81.25% in accuracy, depending on emotion class and method. It’s not possible to directly compare all emotional classes as the results from all methods were not included. For individual users, KNN gave promising results with accuracies ranging from 71.05% to 83.33%. However, KNN was not applied to all emotional states.

Using a multiclass classifier on one single user, resulted in an accuracy of

63.33% for decision tree and Bayesian network, which are the only two classifiers mentioned in this context. The recall rates ranged from 33.3% (class: sadness, method: decision tree) to 100% (class: boredom, method: Bayesian network). Overall, the binary classifier outperformed the multiclass classifier.

3.2.2.3 Using fuzzy logic to classify emotions from keystroke dynamics

In the paper by Shukla and Solanki [2013], two data collection tools are briefly proposed, one for fixed text and one for free text. The fixed text application would allow the user to enter a presented text, and later select one of six emotions. The free text application would prompt the user every 15 minutes for them to enter their emotional state, choosing again from one of six emotions. The emotional classes proposed were: confidence, sadness, nervousness, happiness, tiredness and hesitation. From this data, the features session time (the total time the user used in the system), flight time, dwell time, sequence (digraphs, trigraphs and n-graphs), typing speed (number of keystrokes per minute), error frequency (number of backspace and delete key presses, divided by number of characters in the sequence), pause rate (the time users used to respond to a question) and capitalization rate (number of capital letters in a sequence, divided by the number of characters in that sequence) would be extracted. The paper's author argues that a rule based system for classification, e.g. decision trees, having a discrete set of labels, is not suited for continuous attributes. Thus they propose to use a fuzzy logic system to classify the data. However, no results from this study has been found, and the researcher could not find any progress later than the paper by Solanki and Shukla [2014].

3.2.2.4 Detecting stress from keystroke dynamics

In the study by Hernandez et al. [2014], a pressure-sensitive keyboard and a capacitive mouse was used to detect user's stress level non-intrusively. Data labeling was done by self-reporting, with a sensor to measure electrodermal activity and skin temperature, and with an accelerometer. The research found that stress influenced typing pressure consistently with >79% of the participants (24 participants in total) and there were consistently more mouse contact with 75% of the participants. The article notes that while stress may lead to more muscle activity, and thus alter the typing pressure, other factors may also affect the muscle activity, e.g. excitement and physical activity.

Kořakowska [2016] builds upon the work from Kořakowska [2015] by conducting a preliminary study of how time pressure affect a user's keystroke dynamics. Using statistical analysis, Kořakowska analysed the dataset to find changes in 58 selected features (divided into the sections digraph features, trigraph features,

special digraph features, frequency features and typing speed). The dataset consisted of 36 data samples, each consisting of two parts, one with keystroke dynamics in a normal setting, and one with keystroke dynamics under time pressure. Kołakowska found that about half of the parameters changed significantly when under stress, most of them being digraph and trigraph characteristics and typing speed. The conclusion is drawn on basis of all users, and not individually. Although the study used time pressure as a mean to cause stress, the effect of the time pressure is uncertain, as there was no self-assessment afterwards.

3.2.3 Inferring a programmers performance and experience

Leinonen et al. [2016] wanted to find out if keystroke latency data could explain students performance on exams, and if keystroke dynamics could be used to distinguish between novices and experienced programmers. 223 students from a programming introductory course participated in the study. This is the same data set that was collected by Longi et al. [2015]. Initially, over 10 000 features were extracted from the dataset. After a feature selection process, the researchers were left with 20-50 features to classify. The classification methods used were Bayesian Network and Random Forest classifiers with 10-fold cross-validation. The study found that student's keystroke latencies could be used to explain students' exam performance, and that keystroke latencies could partially be used to distinguish between novices and experienced programmers. On that note, the study found that there was a difference in time spent to move from e.g. the character "i" to "+" between novices and experienced programmers.

3.3 Summary

As noted by Kołakowska [2015], the task of comparing research, specially research on emotion recognition using keystroke dynamics, is made difficult by different researches having different datasets, emotional states and feature sets. However, it was possible to draw some conclusions that could be used to identify a way forward for this research.

In the research by Kołakowska [2015], KNN showed promise for both multi-class and binary classifiers. The algorithm was almost never the best classifier, but on average it performed well, whereas other classifiers performed both well and badly. Longi et al. [2015] also had success with the KNN algorithm when using it to identify programmers. These findings on using KNN in a programming environment makes it relevant for this thesis.

No article on keystroke dynamics and emotion recognition have described the SVM algorithm in use, but it was used by Sung and Cho [2006] to authenticate users with 8%-10% error rate. SVM and KNN share similarities as they are

both instance based algorithms, and use an unknown instance's location in a feature space compared to other already classified instances as a way to classify the instances. Thus it is interesting to see how well SVM fare compared to KNN.

KM et al. [2015] proposed using the pulse as a measure for positive or negative emotions in a multimodal approach. That research, however, did not merge the three proposed data sources. Other researches (Yuksel et al. [2016] and Epp [2010]) also state the pulse's relevance to detect emotional states. . Similar to keystroke dynamics, pulse is also a feature that can be collected non-intrusively, e.g. by using a smartwatch or activity tracker. This makes pulse a good candidate for further study paired with keystroke dynamics.

Chapter 4

Adapt

Keystroke dynamics collection tool

For this research, a JavaScript tutorial web application was developed. Though it was not adaptive, the application was named Adaptive Programming Tutorial, Adapt for short, for its future potential. This chapter explains the application. In the first section, the user interface and interaction between users and the system is presented, followed by the system's architecture.

The application presents the user with exercises that become increasingly more difficult. These exercises were written by the thesis author, inspired by exercises from IT2805 Web Technologies and online JavaScript tutorials, such as Codecademy, W3Schools (<https://www.w3schools.com/js/>) and Tutorialpoint (<https://www.tutorialspoint.com/javascript/>). Keystrokes that the users type in the editor are collected together with their emotional state, as reported by them, and stored in a database.

JavaScript was chosen as the tutorial language because computer science students at NTNU does not learn it during their first year. They do learn Python during their first semester and Java their second semester, however. Thus it was possible to recruit students with programming experience, but who weren't familiar with JavaScript. Recruiting is further discussed in section 5.1.1.

4.1 User interface

The first view users are presented with is an information page about the project, and an link to the application itself. Upon entering the tutorial application, the user is presented with the view seen in figure 4.1 (the webpage is zoomed in at

150%). The view consist of three sections, information about the exercise to the left, the web editor in the center and the console to the right. The editor is the Ace editor (<https://ace.c9.io/>), developed by Cloud9 IDE (<https://c9.io>) and Mozilla (<https://mozilla.org/>), used by sites like Codeacademy. The console outputs the result from the code written in the editor, once the “run” button (bottom center, currently disabled in figure 4.1) is clicked.

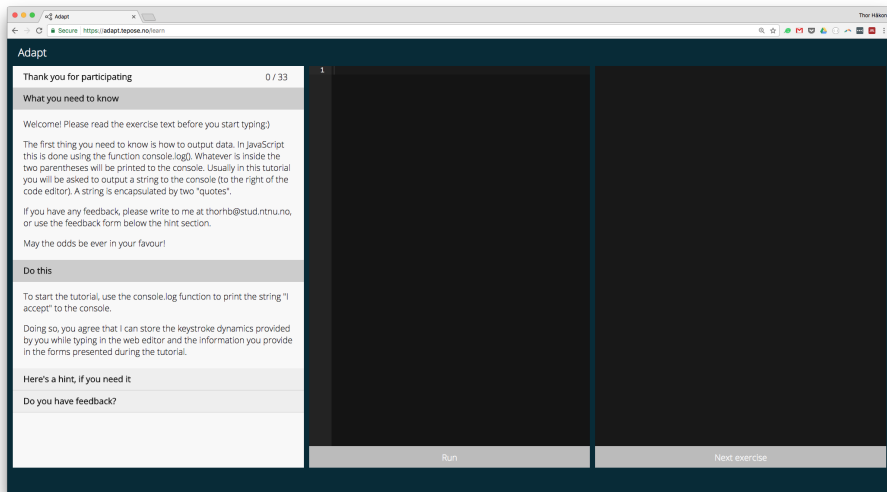


Figure 4.1: Adapt’s initial view

The first exercise asks the user to accept taking part in the study. The application view is identical throughout the tutorial, only the information on the left side changes.

The information section consist of four subsections, curriculum relevant for the exercise (what you need to know), the exercise (do this), a hint if the user is having problems completing the exercise (here’s a hint, if you need it), and a form for the user to report feedback to the researcher (do you have feedback?). The setup is inspired by Codeacademy to make an familiar tutorial to many, as not to take the focus away from the exercises.

The run button is deactivated by default, but becomes active once the user has entered at least one character into the editor. The “next exercise” button is deactivated by default, and will become active once the current exercise is passed.

Users can at any time write a message in the feedback form and send it. This does not affect the exercise, nor will the keystrokes be stored. Once sent, the user is prompted with a “thank you”, giving feedback to the user that the message is

sent.

4.1.1 Collecting user information

Upon entering the application an unique ID will be generated by the backend system, which are then stored in the database. This ID is also stored locally in a browser cookie, so that the user can be identified upon reentering the webpage. The first exercise (exercise 0) contain a short introduction to the research, and asks the user to accept taking part in the project, as seen in figure 4.1. By typing the string `console.log("I accept")` into the editor and running the code, the user confirms to participate in the research. When the user clicks the “next exercise” button the user will be prompted with a form, asking about the user’s gender and experience with programming, see figure 4.2.

May I ask something about you?

What's your gender?*

Male
 Female

What's your experience with programming?*

No experience Intermediate (viderekommen) Expert

What programming languages do you have experience with?*

I have no previous experience with programming
--
 Python
 Java
 JavaScript
 C++
 PHP
 Matlab
 Other languages

How many years have you practiced programming in total?*

How many years before attending university did you practice programming?*

Where did you first learn programming?*

Choose an option ▾

Register Avbryt

Figure 4.2: Adapt’s user information registration form

In order to later categorise users, the user is asked to:

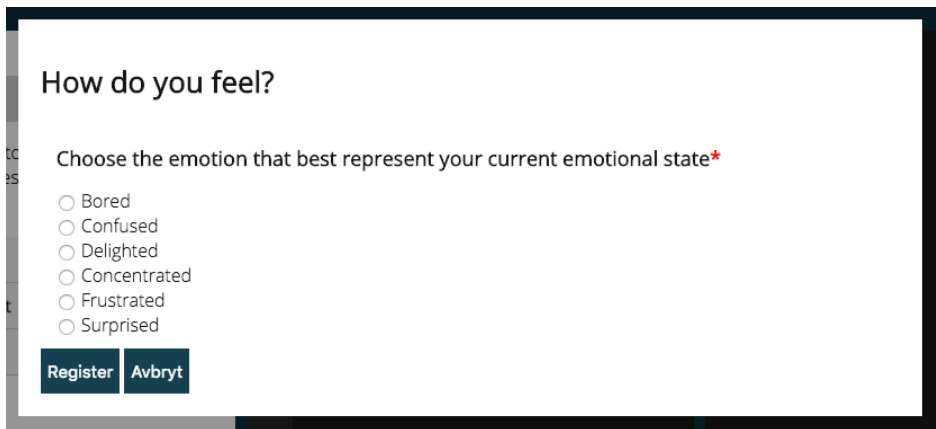
- Describe their experience with programming on a 7-point Likert scale (1 = No experience, 4 = intermediate, and 7 = expert).
- Select which programming languages they have previous experience with (Python, Java, JavaScript, C++, PHP, Matlab or other languages).

- Answer how many years before attending the university they practiced programming (in intervals of 0.5 years).
- Answer how many years in total they have practiced programming (in intervals of 0.5 years).
- Select when they first learned programming (this is my first time, on my own, before middle school, middle school, high school, or university/college)

This information is stored in the database together with the user’s id. The user can choose to close the form by clicking “avbryt” (the researcher forgot to translate this button). However, the form will be presented each time the user clicks the “next exercise” button, until the form is answered and registered.

Each time the user clicks the “run” button, the user is prompted with a form, asking the user to describe their current emotional state. This is a list of radio buttons, each corresponding to one of the specific emotions bored, confused, delighted, concentrated, frustrated and surprised. The form can be seen in figure 4.3. Upon clicking “register” the code is run and the result is made visible in the console. The user can choose to close the form by clicking “avbryt”. Doing so will result in the code not being executed.

Even though the code was executed once the user clicked the “run” button, it is not displayed in the console until the user clicks “register”. Clicking “avbryt” would lead to the result not being displayed. The reason for this was to get the user’s genuine emotional state (Choppin [2000]) and not have the user judge their emotional state based on how well their code performed.



How do you feel?

Choose the emotion that best represent your current emotional state*

Bored

Confused

Delighted

Concentrated

Frustrated

Surprised

Register Avbryt

Figure 4.3: Adapt’s emotion registration form

4.1.2 Exercise handling

After accepting to do the tutorial, the user is given 32 exercises, whereas one of them (exercise 29) is information about the last exercises. The exercise curriculum, text and hint is always given in the left section of the application. When presented with an exercise, the user writes code that should yield the expected results. After doing so, the user must click the “run” button to execute the code.

When the code is evaluated there are three possible outcomes: “correct answer”, “correct code, but wrong answer”, and “coding error”. If the answer is correct, the “next exercise” button will become active, and when clicking it the user will be presented with the next exercise. However, if the user got a wrong answer, they will be presented with an orange error message giving a hint at what might be wrong (see figure 4.4). This message is static, written by the thesis author. As such, it is only a guess at what might be wrong. The result will be visible in the console.

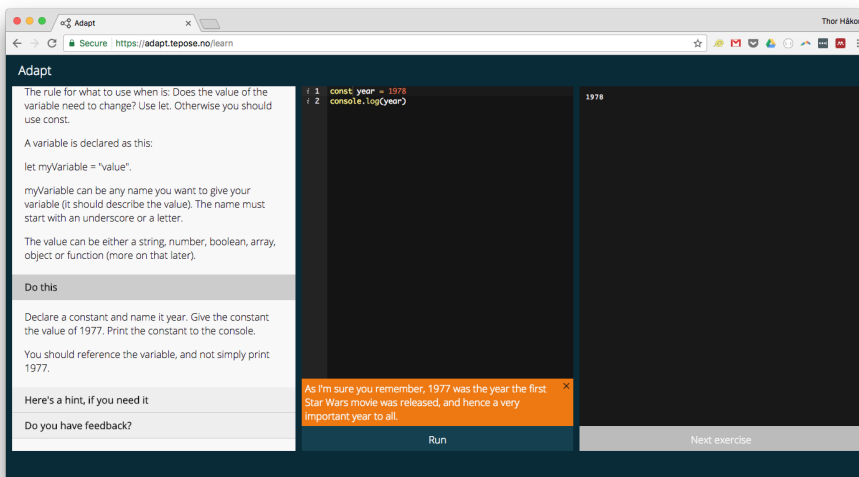


Figure 4.4: Error message display when the result is wrong

Should the code contain errors, making it impossible to run, the user will be presented with an orange error message, stating that there is an error in the code. The error message is provided in the console. See figure 4.5 for an example.

The error message can be exited by clicking the “x” in the top right corner of the message. It will also be closed once the “run” button is clicked again.

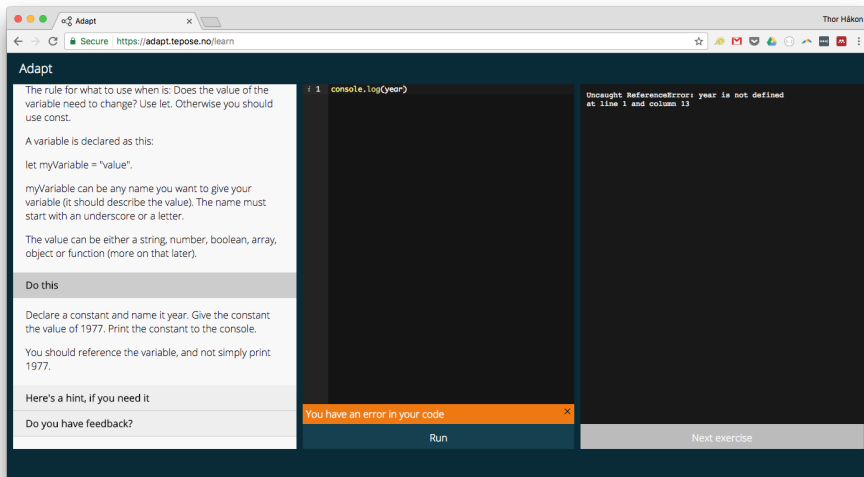


Figure 4.5: Error message when the code contain errors

4.1.3 Inducing emotions

In order to provoke some feelings with the users, as to get a better class distribution, two emotion inducing features were implemented. These are:

- Random removal of text
- 4 minute timer

The random text removal feature was implemented after feedback from the supervisor, and the timer was inspired by Kołakowska [2016]. The random text removal might happen any time, but the timer only occurred on selected exercises.

For each character typed, there is a one in 300 chance that the editor will be reset, i.e. all written text will be removed from the editor and the user will need to start again. The first exercises are fairly short, thus the chance of removal is small. On the latter exercises, however, the chance increases as they require more code. Once removed, the user is not given any information on what just happened. The number 300 was chosen after some test runs, and feedback from some early participants, so that it didn't occur too often, nor too seldom.

The other feature, the timer, was visible at the top of the editor on selected exercises (16, 17, 30, 31 and 32). The timer is not announced with exercises 16 and 17, but it is announced before the user starts exercises 30, 31 and 32. The

timer starts at 4 minutes, which is an arbitrary number. When it enters 1 minute, the color changes to orange to signal that the end is near. Upon reaching 0, the color turns red. The timer's three stages can be viewed in figure 4.6.

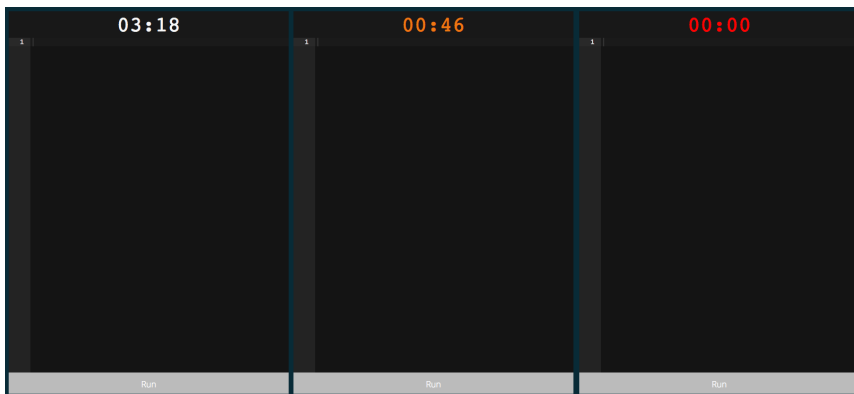


Figure 4.6: The timers three stages.

When one or both of the emotion inducing features occurred, the user would be prompted to give feedback on whether they were affected by them. The user would be asked to answer either yes or no. The user would also be asked to give a written explanation on how they were affected. Due to faulty programming logic, however, these text fields weren't required to answer.

Should both features occur, the user would be asked to give feedback on both. The prompt would come once the user clicked the "run" button, together with the emotion selection. The prompt for random removal of text and the timer can be seen in figure 4.7. As can be seen, the question for further explanation on how the feature affected the users changed depending on the user's answer (yes or no).

4.1.4 Interaction with the editor

With the provided code editor, it wasn't possible to paste code (or any other text), it is also wasn't possible to redo an action. The decision to restrict the users this way came after a test user got bored writing the same code over and over again, and started pasting (CMD+V on macOS and CTRL+V on Windows) the code into the editor. This resulted in the session containing only a few keystrokes, rather than keystrokes for the complete code. Similarly, when the random text removal occurs, one user found that it was possible to redo (CMD+Z on macOS and CTRL+Z on Windows) the last change, resetting the editor to the pre-removal state. This also reduced the number of collected keystrokes.

Figure 4.7: Feedback form for emotion inducing features.

After the disabling, when trying to paste, users would be prompted with the error message “Pasting code have a negative impact on the recorded data and have thus been disabled”. No information was given when the user tried to use redo, it would simply not work. These features were set in the Ace Editor’s configuration file.

4.2 System architecture

To understand the architecture, it is important to define some important terms. These terms are used by the researcher to define parts of the system, and are defined by the researcher.

- **Key object:** Each interaction between the user and the keyboard is stored in an object consisting of the values: Key code (a unique code for one specific key), key press timestamp and key release timestamp. Every timestamp is expressed in milliseconds.
- **Keylog:** Every key object is stored sequentially in a list by when the keys were pressed.
- **Session:** A session takes place between two code runs (clicking the run button), or from the user first enters the page and the first code run. A session object consist of the keylog for that period, exercise id, user id, the user’s emotional state as stated by the user, the code written, the result and information on how the emotion inducing features affected the user.

The frontend application is written in HTML, CSS and plain JavaScript, and the backend is written in Python and the microframework Flask (flask.pocoo.org), while the database is the document database MongoDB, developed by MongoDB,

Inc. (<https://www.mongodb.com/>). The backend serves the client application through an API. The web application does not reload when a new exercise is presented, but the keylog is reset.

4.2.1 Key logging

At the heart of the application is the collection of keystroke data. When users type on the keyboard, the browser fires key events. These events are captured by the application and stored in a keylog list. Each key event can be one of three types: keydown, keypress or keyup. For this research, the keydown and keyup events are interesting as they represent when a key is pressed down and released, respectively. Each key event contain information about the key's unique identifier and a timestamp for when the event occurred.

Each time a keydown event is registered, the key code and timestamp is registered and stored in the keylog list. If a keyup event is registered, the list will be looped through, starting from the end, until a key object is found with the same key code, i.e. the last key object with the same key code. This approach is inspired by the proof-of-concept web authentication with keystroke dynamic, developed by Young [2012]. The information flow can be seen in figure 4.8.

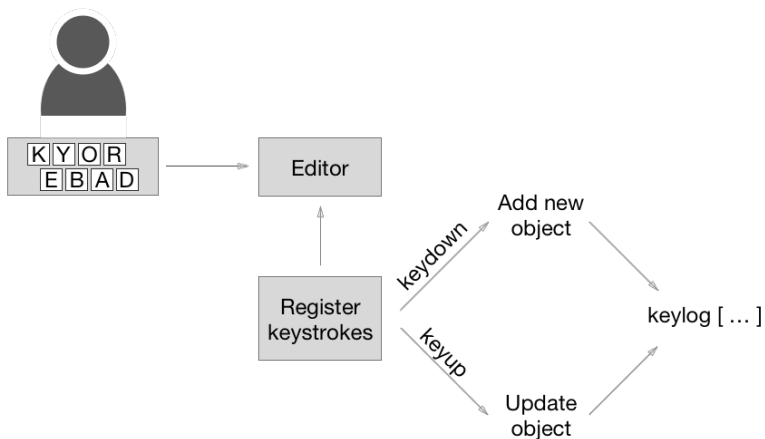


Figure 4.8: Keystroke capture flow

When the user type characters in the editor, the application records each event. If it is a keydown event, a new key object is stored in the keylog. If the event is a keyup event, the corresponding key object is updated with a time up value.

If the user experienced a random text removal, a key object would be added to the log with the key code 256 (this key code is not used by any of the keyboard's keys), to note which key objects occur before and after the text removal. Every key object that occurred after the last time the "run" button was clicked, is stored in the keylog, even when the text in the editor was removed. Also the keys "backspace", "arrow left", "arrow up", "arrow right", "arrow down" and "delete" were stored in the keylog. These keys do not have a keyup event, thus the keyup value is the same as the keydown value. Should a key be held down over a period of time, it will generate several consecutive keydown events. These events are ignored. Only when there is a keyup event in between two keys of the same type will a new key object be created.

When the user click the "run" button, the keylog is added to a session object together with the user id, the content of the editor, the result from the code run, number of backspace and delete keys in the keylog, the exercise id and a flag set to true if the random text removal occurred. After clicking the "run" button, the user is prompted with the form shown in figure 4.3 (or figure 4.7 if one of the emotion inducing features occurred). After clicking "register", information from this form (emotional state and how the features affected the user) is added to the session object. Then that object is transferred to the database through the API. Before it is added to the database, a timestamp is added to include information on when the object was created. If the storage is successful the keylog list is reset, so that only new keystrokes are stored together with the session object. The session object model and example can be viewed in table 4.1.

4.2.2 Users

As mentioned in subsection 4.1.1, once the user enters the application, a user object is stored in the database with an user ID. The same user ID is stored in a browser cookie, as to track users across browser sessions. The ID is created by MongoDB, and is guaranteed to be unique.

After the user has run the code `console.log("I accept")` as part of exercise 0, the user is prompted with a form, asking the user to give information about gender and programming experience, see figure 4.2. This information is stored in the user object in the database, making it possible to classify users later on, and e.g. compare only the feature vectors of those that do not have previous experience with JavaScript. In addition to the information given in the prompt, the user agent is stored to distinguish those that e.g. have used a smartphone from those using a laptop or desktop. Also, there is a timestamp for when the user was created. There is no information on the keyboard used, so it is not possible to see if users have changed keyboard during their testing. It is also not possible to track users across computers or browsers, as the user ID is set on a

Field	Data type	Description	Example
<code>.id</code>	ObjectId (MongoDB data type)	The object's ID set by MongoDB	ObjectId("58e4c3210b859a646c6b6db7")
<code>user_id</code>	String	The user's ID	"58e4c1f20b859a646c6b6db2"
<code>emotion</code>	String	The user's self-described emotional state	"confused"
<code>timestamp</code>	Datetime	Timestamp for when the session was stored	2017-04-05T10:12:49.484358+00:00
<code>result</code>	String	The result from the code run	"42.4"
<code>code</code>	String	Editor's content	"console.log(42.4)"
<code>exercise_id</code>	Integer	Exercise's ID number	4
<code>number_of_misses</code>	Integer	Number of misses during the session	2
<code>keylog</code>	List of objects	The keylog (represented here by the first and last object)	[{"key_code" : 67, "time_down" : 266019.85000000003, "time_up" : 266142.895}, ..., {"key_code" : 57, "time_down" : 277932.92000000004, "time_up" : 278040.24000000005}]
<code>timer_affected</code>	Integer	4 minute timer stress feature 0: Timer didn't occur 1: Timer didn't affect 2: Timer did affect	0
<code>timer_affected_reason</code>	String	Description by the user how the timer affected them	""
<code>was_reset</code>	Boolean	A flag set to true if the editor's content was randomly removed	true
<code>removal_affected</code>	Integer	Random removal of text stress feature 0: Removal didn't occur 1: Removal didn't affect 2: Removal did affect	2
<code>removal_affected_reason</code>	String	Description by the user how the random text removal affected them	"Usikker på om jeg hadde trykket noe feil på tastaturet. Litt paff kanskje?"
<code>pulse</code>	Float	User's average pulse for that session	58.7
<code>min_pulse</code>	Float	User's minimum pulse	58
<code>max_pulse</code>	Float	User's maximum pulse	60

Table 4.1: Adapt's session model with example

browser basis.

Four fields were added to the user object that ended up not being used: username, email, password and completed. These fields are only present for potential future development. The user model can be seen in table 4.2. Each field is explained by data type and a description, and an example is given for each field.

Field	Data type	Description	Example
.id	ObjectId (MongoDB data type)	The user's ID, created by MongoDB	ObjectId("58cef4460b859a30c46fab7a")
user_agent	String	Information about the user's browser, so that e.g. mobile phones can be excluded from the dataset	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/57.0.2987.133 Safari/537.36"
is_admin	Boolean	A flag stating if the user is an admin or not. Admins can add and change exercises	false
created	Datetime	Time and date for when the user was created	"2017-03-28T13:15:45.101552+00:00"
gender	String	Can be either male or female	female
experience	Integer	Integer from 1 to 7, describing the user's experience	2
languages	Object with Integers and one String	Six predefined languages, which can have the value 1 (does not know) or 2 (do know), and one string named other, where user's can add other languages	"python" : 2 "php" : 1 "javascript" : 2 "java" : 2 "cplusplus" : 1 "matlab" : 1
years_of_experience	Float	The total number of years with programming experience, in intervals of 0.5 years as to accommodate for academic terms	0.5
years_before_university	Float	The total number of years with programming experience before attending the university, in intervals of 0.5 years as to accommodate for academic terms	0
first_learned	String	Description of when the user first learned programming, this can be one of: this is my first time, on my own, before middle school, middle school, high school, college/university	"university"
current_exercise	Integer	The exercise id of the last exercise the user started, but has yet not completed	1
username (not in use)	String	Could be used to track users across browsers and computers	"nousername"
email (not in use)	String	Could be used with user registration	"no@email.com"
password (not in use)	String	Could be used with user registration	"doesntexist"
completed (not in use)	Boolean	A flag stating if the user have completed the tutorial. If true, the user could e.g. jump to any exercise	false

Table 4.2: Adapt's user model with example

4.2.3 Exercises

After the initial information in exercise 0, users are presented with 32 exercises with increasing difficulty. The subjects are, in order of presentation: basic data types (string, int, boolean), quotes, concatenation, decimals, arithmetic,

comparing data types, variables and constants, arrays, loops, if/else statements, functions, and lastly three challenges.

Each exercise object consist of the values: title, and ID, curriculum, exercise text, a hint, an error message, the expected result, a timer flag (if set, the exercise is timed), and editor content. Editor content is not used in any of the exercises, but allow for some code to be visible for the user, e.g. if the user should be provided with skeleton code. The reason for not using the “editor code” field is that the users should write as much code as possible, giving more data to analyse.

The general exercise model, and with exercise 6 as an example, is presented in table 4.3. The complete set of exercises can be found in appendix A.

Field	Data type	Description	Example
_id	ObjectId (MongoDB data type)	The object’s database reference	ObjectId(“58de848a0b859a677433b98d”)
title	String	Exercise’s title	(Arithmetic + 5) / 2
exercise_id	Integer	Exercise’s order number	6
version	Integer	Exercise’s version number	1
created	Datetime	Timestamp for when the exercise was created	2017-03-31T16:32:10.237608+00:00
curriculum	String	Curriculum	Arithmetic in JavaScript is controlled the same way as regular arithmetic. Multiplication and division have a higher precedence than addition and subtraction. As with regular math, you can use parentheses to control when a calculation is made, e.g. if you want the addition to happen before multiplication.
text	String	Exercise text	Use parentheses on the calculation $4 + 2 * 100$, so that the answer becomes 600 and not 204. Print the answer to the console.
hint	String	Hint	The order of calculation is as follows: – Calculate parentheses – Multiplication and division, calculated left to right – Addition and subtraction, calculated left to right.
error_message	String	Error message	Did you encapsulate $4 + 2$ in parentheses?
result	String	Expected result	600
timed	Boolean	A flag that, if set to true, will display the timer	false
editor (not in use)	String	Content of the editor	

Table 4.3: Adapt’s exercise model

Once the user clicks the “run” button, the code is run in a web worker. A web worker is a HTML5 feature that allow for safely execution of JavaScript in the background, without affecting the performance of the web page, and does not influence the client side JavaScript.

After the code execution, the result is sent to the backend together with the session object for evaluation. If the user’s result is equal to the exercise’s expected result, the user is allowed to proceed. When comparing answers, line breaks and spaces are removed, all characters are made lower case, and all double quotes are converted to single quotes. This lets the answers be a bit flexible, rather than strictly compare the two strings which would have made the string “I accept” not equal to “i accept”. There is no testing of the actual code, so user’s can write “console.log(600)” on exercise 6 and get a pass. However, the code is stored together with the session, so it is possible to implement some checks later on.

4.2.4 Backend API

The server side code receives data from, and sends data to, the client, possibly adding some data to it along the way. The API also handles sending data to the MongoDB database. The API endpoints used in Adapt are described in table 4.4.

Endpoint	Arguments	Description
add_exercises		Adds a new exercise to the database. Exercise_id is create by the MongoDB
add_feedback		Adds a feedback to the database (contains exercise_id, user_id and feedback text)
add_session		Adds a session to the database (user_id is collected from the cookie)
get_exercises	user_id	Returns the latest version of the next exercise for a given user
get_exercises_by_id	exercise_id	Returns the latest version of a specific exercise
update_exercises	exercise_id	Updates a specified exercise, creating a new version
update_users		Adds user information to an already existing user (gets user_id from the cookie)
post_users		Creates a new user in the database

Table 4.4: Adapt’s API endpoints

Chapter 5

Data collection and experiments

In this chapter, the data collection and experiment plans and their executions, are described. As a part of this research it was necessary to collect a dataset that could be used to infer the chosen emotional states from keystroke dynamics gathered in a programming environment. Data collection was done using the application Adapt, which collected data from users. The experiments aim at classifying the dataset, using KNN and SVM with different hyperparameters and approaches, in order to find the best classifier.

This chapter consists of four sections, first the data collection plan is presented, followed by a section on how it was set up and executed. Then the experiments are explained in detail. Lastly an overview over the participants is given.

5.1 Data collection plan

Similarly to Epp [2010] and Kołakowska [2015], data collection for this research was done using observation and measurement, where quantitative data was collected for later analysis (Oates [2005]). Adapt, the data collection tool described in chapter 4, observed participant's keystrokes automatically. This made it possible to collect large amounts of data, without the researcher's presence. The data collection sessions¹ which the researcher did not observe, are called unsupervised sessions in this thesis. To collect the pulse it was necessary to ask some participants to take part in a supervised data collection session, where the researcher

¹Data collection sessions must not be confused with the sessions between two code runs.

observed the participant's pulse while they completed the tutorial. In both data collection sessions participants were asked to self-report their emotional state in the questionnaire presented in figure 4.3.

5.1.1 Participants

As JavaScript was the chosen language, participants without previous experience with the language were preferred. The researcher hypothesised that these students would have a greater variance in how they reacted to the exercises than those familiar with the language. Additionally, participants without experience with JavaScript might find the exercises more challenging than those with experience. However, it was desirable to have participants with some programming experience, so that they didn't need to learn the very basics of programming. As bachelor and master students in informatics (the researcher's study program) have experience with JavaScript through the course IT2805 Web technologies, first year students from the five years integrated master's degree in computer science were recruited (they do not attend IT2805).

These students learned Python in an introductory programming course during their first semester. During the data collection phase, they are in their second semester where they are learning Java in an introductory course on object-oriented programming. Thus, the computer science students are familiar with programming, but are not necessarily proficient.

As reported by both Epp [2010] and Kołakowska [2015], their datasets were limited in size due to a small number of participants, 12 and 9 respectively. Hence, this research should aim to increase the number of participants, with no upper limit. Recruiting was done through the researcher's network at NTNU and during a presentation in TDT4100 object-oriented programming.

5.1.2 Data collection

To gather data, participants were introduced to the tutorial application described in chapter 4. Completing the tutorial in one sitting takes about 40 to 60 minutes. By using this tutorial, participant's keystroke data was collected and stored in a database for later analysis. If the participants choose to come into the lab and go through the tutorial, their pulse data would also be stored in the database.

Participants in the study by Epp [2010] used an average of four weeks to collect enough data, while participants in the study by Kołakowska [2015] used from two weeks to three months, and participants in the the study by Longi et al. [2015] used 7 weeks to collect data, due to the programming course's length. Following these researches, and due to the duration of this master's thesis, data collection was set to last for four weeks, but it was possible to also gather data after this.

The collected keystroke dynamics are discrete quantitative data by nature, as each event occur between two specific timestamps. Pulse, on the other hand, is continuous, but was in this research stored as discrete values, measured in 10 second intervals.

In this research there were two types of data collection settings: Participants could do the tutorial without supervision from the researcher (unsupervised data collection), or participants could do the tutorial with supervision from the researcher (supervised data collection). Unsupervised data collection was used in order to collect large amounts of keystroke data in a time efficient and scalable manner. This way, there was no limit to how many could participate, as it demanded no resources from the researcher. Supervised experiments were used to gather pulse data in addition to the keystroke data. This demanded, however, that the researcher was present. Participants could take part in both data collection types, but preferably first in the supervised data collection to capture their initial response to the exercises together with their pulse data.

5.1.2.1 Unsupervised data collection

With the unsupervised data collection, the researcher had to trust that participants had good intentions and that they did not try to deliberately apply the incorrect label. This approach is similar to Kołakowska [2015]. Allowing participants to do the tutorial on their own time, and in a place that is natural to them, would give more authentic emotions. As stated by Kołakowska [2015] this is important to get good data. As the number of exercises were limited, participants were encouraged to retake the tutorial when they experienced different moods or emotional states, thus generating a greater number of instances for each class.

5.1.2.2 Supervised data collection

During this research, there was no time to develop an automatic collection of pulse data, hence it was not possible to collect pulse during the unsupervised experiments. In order to capture this data, participants were asked to do the tutorial while supervised, and with a non-intrusive pulse detecting device. This information was updated live, displaying the data on the researcher's phone, so that the pulse measures could be written down at 10 second intervals. 10 seconds is an arbitrary number, allowing for longer exercises to have a manageable number of entries.

A python application that uses the laptop's integrated webcam to track a user's pulse based on changes in their skin colour, by Hearn [2013], was tested as a non-intrusive method to collect participant's pulse. However, this application required participants to sit perfectly still while doing the exercises. As that was an unlikely scenario, a smartwatch or activity tracker was preferred. Such a device

was also preferred rather than a dedicated pulse monitor as the smartwatches are more likely to be used in a natural setting, e.g. while studying. These devices are owned by many, while a dedicated pulse monitor is mostly owned by those who are serious about their workouts.

When participants arrived, they were given time to calm down and ready themselves, as to limit influence from outside the controlled environment. The researcher started with some small talk, and practical information about the research as seen in the list below:

- Talk about the research’s goal
- Introduce the participants to their task
- Let them know that they aren’t the ones who are being tested, they are here only to provide data
- Introduce the data collection tool
- Tell the participants that they should answer honestly, and that the researcher would not get offended should the participant answer “bored”

During the experiment, the participant’s laptop’s screen and the researcher’s phone (where the pulse was displayed) were filmed so that it would be possible to go through the data later, controlling and correcting pulse values where there was a mismatch between the number of sessions and the researcher’s notes. It was not necessary to film the participant’s face, only the screen to see when the “run” button was clicked, indicating the end of a session.

After the experiment, the researcher would debrief the participant to get insight into possible improvements, how the participant found the tutorial and application interface, and how they were affected by emotion inducing features.

5.2 Data collection setup

The web tutorial described in chapter 4 was a crucial part of the data collection setup. As the application should not be a limiting factor for the participants, nor distract them from the exercises, it was decided to make it similar to Codeacademy’s tutorial. User interaction, layout and flow in the application was discussed with an fourth year interaction design student at NTNU.

Participants were asked to use their own computer and keyboard while doing the tutorial, as users type differently on a keyboard they are unfamiliar with (Teh et al. [2013]). The researcher could not control this during the unsupervised sessions, but participants taking part in the supervised sessions were asked to bring their own computer.

5.2.1 Supervised data collection setup

During the supervised sessions, participants were equipped with a Fitbit Charge 2 activity tracker, Fitbit [2016], on their non-dominant arm (as per instructions). The Fitbit was chosen over other smartwatches and activity trackers because of Fitbits widespread use, and possibility for later integration against an application to automatically register pulse data on both Android and iOS devices. Salazar et al. [2017] found that the Fitbit Charge 2 provided adequate values for recreational use, but not for serious workouts. This led to the decision that the Fitbit was good enough for this research, and also fitted well within the goal of being non-intrusive.

The phone that displayed the pulse had to be positioned outside the participant's view so that the participant did not focus on their own pulse. This was first done in later data collection sessions, after feedback from some participants. The Fitbit was connected to the phone via Bluetooth to display the pulse live. The camera did not capture any footage except for when the participant did the tutorial, so that no passwords or other sensitive data were recorded. A screenshot from the supervised data collection setup can be seen in figure 5.1.

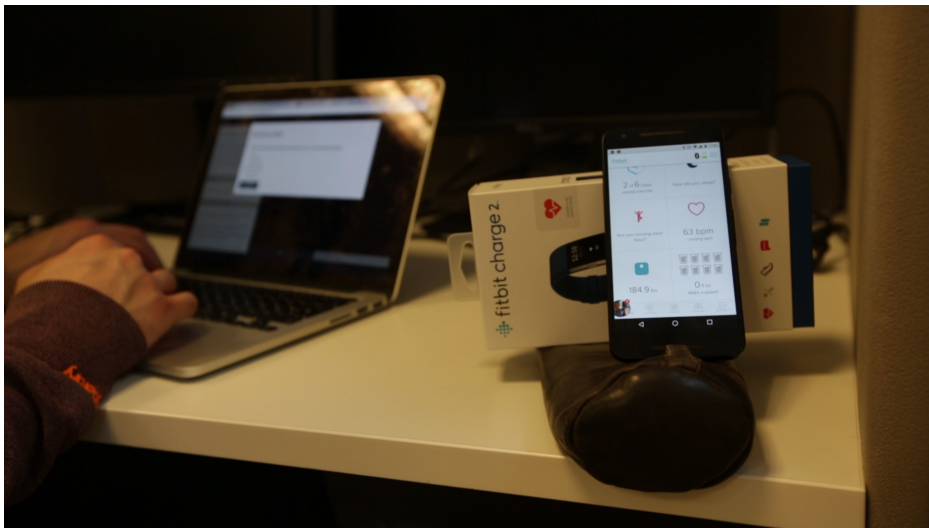


Figure 5.1: Screenshot from the supervised data collection session.

5.2.2 Personal information handling

No written personal information (name, address, e-mail, birthday) was collected that could link recorded data to one specific person. However, as reported by Longi et al. [2015], it is potentially possible to infer a person's identity from their keystroke dynamics. More importantly for this study was the filming of some participants, and potentially so using the webcam (as described in section 5.1.2.2). The collection of such data could be used to detect people's identity.

Thus, it was necessary to register the research with the Norwegian Centre for Research Data (NSD) and apply for being able to collect data. The application can be seen in its entirety in appendix B. A part of the application was the information letter given to participants in the supervised data collection. This information letter can be seen in appendix C. The response from NSD can be seen in appendix D.

The video files of participants' pulse were stored locally on the researcher's computer, and deleted once the pulse was added to the sessions objects. No video was taken using the webcam, and no footage showed the participant's face.

5.3 Experiments

In this section the collected data and the classification tool developed for this research is explained, together with all the necessary feature vector creation methods. This tool, which in all fairness is a script, extracts and derives features from the collected dataset, and apply them in the selected classification methods.

5.3.1 Nature of the dataset

To better understand the feature extraction and classification processes, it is necessary to understand how the dataset looks like, and what it consist of.

Each user is stored as an object with the fields described in table 4.2. The important fields to notice, however, are: `_id`, `experience`, `languages` and `years_of_experience`. These fields make it possible to classify users against similar users, e.g. to differentiate between novice and expert programmers, as these might have different traits in their typing rhythm.

Each session (the data collected between two code runs) is stored as an object with the fields described in table 4.1. The important fields to notice here are: `emotion`, `keylog`, `pulse`, `min_pulse` and `max_pulse`. These fields are, together with fields from the user object, used to create the feature vector used during classification.

The keylog list in the session object was used to create the main bulk of features in the vector. From here, it was possible to derive a number of features,

which describe the relationship between two or three keys, and how each key is pressed. The keylog is ordered by when each key was pressed down, meaning that if key A, B and C was pressed down at the times $t+1$, $t+4$ and $t+2$ respectively, they would be ordered A, C, B. The release time is not considered during ordering. As such it is possible to get negative values when calculating the flight time between two keys (the time between key A is released and key B is pressed).

Classifying two vectors require that each pair of elements in the two vectors represent the same feature. When participants type, it is possible that they make mistakes, or that they move the cursor to another part of the text. As pointed out by Epp [2010], it is difficult to track corrections as users can use both the mouse and arrow keys to navigate the editor. E.g. a user can use the up arrow to go to another line, but as line breaks occur automatically dependent on the browser's width, the arrow up key might take two users to a different part of the text. In order to compare two vectors that make up the same word, but with different sets of keys, some actions must be taken. These actions are explored in section 5.3.2.1.

5.3.2 Data classification tool

The data analysis tool was written in Python, with classification methods from the data analysis library Scikit Learn by Pedregosa et al. [2011]. The tool takes user and session objects as inputs, creates feature vectors, and classify the vectors using the classification methods KNN and SVM.

As mentioned, it is only meaningful to compare two instances of the same sequence of characters, so that you directly can compare the difference between two equal features. In order to find these sequences, each keylog is compared to a pattern defined by the author. A pattern is a sequence of keys that make up a word or string of characters. Each pattern is an object consisting of the fields "text" and "key_codes", where "text" is the pattern in plaintext, and "key_codes" is a list with the key codes. E.g. key codes for the pattern "let" (a JavaScript keyword) is "[76, 69, 84]". The object can be seen in table 5.1.

Table 5.1: Pattern used to recognise sequence of characters

Field	Data type	Description	Example
text	String	The pattern in plaintext	true
key_codes	List of integers	Each list element is a character's key code. The list is ordered by the key's appearance in the word. One key can occur several times.	[84, 82, 85, 69]

The classification process can be divided into three main steps: feature vector construction, feature vector processing, and classification. Only valid users (users

who have submitted the user information form seen in figure 4.2) are used during the classification. Session instances belonging to non-valid users are disregarded.

5.3.2.1 Feature vector construction

The first step is to construct the feature vectors. This phase is adapted from the research done by Epp [2010], and later Kołakowska [2015]. Not all steps from the two articles were clear, only which features to include, but not how they were applied to the data samples. Thus, some assumptions and adjustments were made.

The construction is itself divided into smaller substeps: keylog cleanup, pattern recognition, and feature calculation. These steps are executed on all session objects that have a keylog with more than one element. The shortest pattern is the two character keyword “if” ([73, 70]), and since digraphs and trigraphs features make up a large portion of the feature vector, keylogs with less than two characters are excluded.

Keylog cleanup Every keylog (that is two characters or longer) is first cleaned up in order to recognise subsequences, i.e. find the sequence of keys that makes up a pattern. One keylog may e.g. contain the character sequence [82, 69, 84, 85, 84, 8, 82, 78] (r, e, t, u, t, backspace, r, n). This should be stored as an instance of the pattern “return”, but because there is a “t” and a “backspace” too many, the pattern will not identify the keyword without first removing the extra keys.

The list of key objects in the keylog is looped through, and for each key, the key object is added to a clean list (a list without certain characters). If the loop finds any of the keys backspace (key code 8), delete (32), arrow left (37), arrow up (38), arrow right (39) or arrow down (40), they will not be added to the clean list. This also is the case if the loop finds a key object with the key code 256, which indicates that the editor was randomly reset.

However, if the loop finds the backspace key (8), it naively assumes that the previous character was removed. The last key added to the clean list, before the backspace was found, will then also be removed. This approach will handle the scenario presented in the previous paragraph, with the keylog [82, 69, 84, 85, 84, 8, 82, 78]. However, it would not handle instances where the mouse has moved the cursor, or the cursor has been moved with the arrow keys.

Using this approach for the pattern “console.log(“ results in 530 valid instances, rather than 469 when the cleanup was not done.

Pattern recognition The next step is to recognise patterns within a (clean) keylog. A keylog may consist of one, or any positive number, key objects.

Pattern recognition is done with a method that takes a keylog and a pattern as parameters. The keylog is looped through, and if the current key element matches the first element in the pattern, all following characters in the keylog is checked against their respective element in the pattern. In case of a match, the first and last element's index are appended to a list, and the loop continues. It is thus possible to find several matches within the same keylog. E.g. this is the instance in exercise 26 where participants are asked to type both an "if" statement, and an "else if" statement, generating two subsequences for the keyword "if".

The list with the subsequence's start and end indices are then used to extract the correct key objects from the keylog. This reduced keylog, i.e. only the key objects corresponding to a certain pattern, is used to construct the feature vector.

Feature selection and construction For each reduced keylog, a feature vector is constructed. The vector's length varies with different patterns. Similar for all feature vectors, however, are the 8 first features. The first 7 of these contain information about the user (as collected from the user object). The 8th feature is the vector's class. These features weren't used during classification, but rather before, when the researcher decided which vectors to include in that specific classification. E.g. it could be desirable to exclude all vectors of the classes bored and surprised. The remaining features are three pulse features, used for vectors where the pulse was measured, and derived features from the keystroke data.

Feature vectors based on the pattern "console.log(" have 153 derived features (156 for pulse vectors). These features are normalised, so that units that would otherwise be different (milliseconds and beats per minute) can be expressed within the same range. The derived features are found by looping through the reduced keylog, calculating the features described in table 5.2.

The features in table 5.2 are selected on the basis of research done by Epp [2010] and Kołakowska [2015]. However, the features "number of events in the digraph" and "number of events in the trigraph" were not taken into account as each key object in this research is stored with both key press and key release, meaning that each digraph would always consist of six events and each trigraph always consist of nine events. In addition, single key dwell times are not included, as they are in Kołakowska [2015], since these features are calculated in both digraphs and trigraphs. The relationship between key events, dwell time and flight time can be seen in figure 5.2.

The average pulse feature is calculated by finding the mean of all pulse measures for a given session (all keystrokes between two code runs). The max and min pulse values are the highest and lowest pulse measured during the session.

Average dwell time was calculated finding the mean of each key's dwell time in the reduced keylog. Key objects that are not a part of the reduced keylog are not taken into account. Average flight time is calculated finding the mean of the

Table 5.2: Features used in this thesis. Note that the pulse features are only used when they have been collected.

Applies to	Feature description
Entire session when pulse was collected	Average pulse
	Max pulse
	Min pulse
Entire session	Number of misses
	Average dwell time
	Average flight time
Digraphs	Time between pressing the first and the second key in a digraph
	Dwell time for the first key
	Flight time between the first and second key in a digraph
	Dwell time for the second key
	Digraph duration (time between pressing the first key and releasing the second key)
Trigraphs	Time between pressing the first and second key in a trigraph
	Dwell time for the first key
	Flight time between the first and second key
	Time between pressing the second and third key in a trigraph
	Dwell time for the second key
	Flight time between the second and third key
	Dwell time for the third key
	Trigraph duration (time between pressing the first key and releasing the third key)

flight times between all key objects in the reduced keylog. Average keystrokes per minute is the number of key objects in the reduced keylog divided by the duration of the reduced keylog (the last key object’s release time minus the first key object’s press time).

The number of mistakes is the sum of all backspace (key code 8) and delete (key code 32) keys during a session (the full keylog).

Digraph and trigraph features are calculated on all pairs and triples of key objects. This means that for a sequence of characters [70, 65, 76, 83, 69], which represents the keyword “false”, digraph features are calculated on the character pairs 70 and 65; 65 and 76; 76 and 83; and lastly 83 and 69. Trigraph features are calculated on the character triplets 70, 65 and 76; 65, 76 and 83; and lastly 76, 83 and 69. Note that the last digraph and trigraph are those that have the sequence’s last element as their last element.

For the sequence mentioned above, the feature “flight time between the first and second key in a digraph” is calculated by subtracting the second element’s

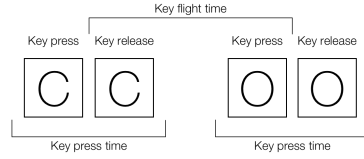


Figure 5.2: Relationship between key press time and key flight time for two keystrokes, adapted from Shukla and Solanki [2013].

press time from the first element’s release time. I.e. for a digraph pair (a, b), the flight time between these two elements are

$$press_time(b) - release_time(a)$$

For the trigraph feature “tripgraph duration”, the duration is calculated by subtracting the last triple element’s release time from the first element’s press time. I.e. for a trigraph triple (c, d, e), the trigraph duration is

$$release_time(e) - press_time(c)$$

As each feature is calculated, they are also normalised between the range 0 and 1. The normalisation method is

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

However, values can get higher than 1, no feature is limited to 0 or 1.

For the pulse features, the min value was 45 and the max value was 90. The number_of_misses feature had a min value of 0 and a max value of 100. Average dwell and flight time had a min value of 0 and a max value of 150. The duration had a min value of 0 and a max value of 600. All digraph and trigraph values had a min value of -100 and a max value of 500. The min value was negative due to the nature of how users type. For some features it was possible to get a negative value, e.g. for the flight time when a key was pressed before the previous key was released. These min and max values were set based on observations made in the dataset.

5.3.2.2 Feature vector processing

There are some limitations in the feature vectors as some keystrokes were removed from the keylog, and that the feature extraction did not consider participants

taking a break from typing during one session, as pointed out by Epp [2010]. Because of this limitation, vectors with features that had a value greater than 40 times the upper bound set in the normalisation function, i.e. the feature's value was 40 or greater, were removed from the dataset as a step to remove outliers. This process reduced the dataset for the pattern "console.log(" from 530 to 521 instances.

This was a limited preprocessing before the classification, and many other steps could have been taken, such as feature selection, though time did not allow for this.

5.3.2.3 Classification

The last step was to classify the feature vectors. This was done using the classification methods KNN and SVM.

The first thing that happened in the classification step was that the vectors were split into two lists, one with the feature vector itself, and one list with the class label (the vector's emotional state). Data used to categorise the vectors (user_id, experience, years_of_experience, years_before_university, first_learned, knows_javascript and gender) was excluded from the vector, as it held no data on the user's emotional state.

The vector and label lists were then randomly divided into a training set and a validation set with 2/3 of the vectors in the training set and 1/3 in the validation set, as is practice. In order to get the most out of the limited data set, k-fold cross-validation was used (Witten et al. [2016]).

Because the number of instances in each class varied greatly, as was the case with Epp [2010], undersampling was used on the dataset to create an uniform distributed dataset (Beckmann et al. [2015]). In addition, it was possible to remove classes from the classification, e.g. not use instances of the class surprised at all. This allowed for a multiclass classifier on a subset of the classes.

In addition to the six predefined classes, a seventh class "other" was added for use in the binary classifier. This way instances that were not of a certain class, e.g. "bored", could be labeled as "other", and "bored" could be classified against all other classes in a binary classifier. Epp [2010] reported that the binary classifier yielded the most promising results, consequently it should also be included in this study.

When choosing to classify vectors where the participant's pulse was measured (vectors from the supervised sessions), only sessions registered with a pulse greater than zero (i.e. the pulse was measured) were used. This, however, lead to a far smaller dataset. Similarly, when feature vectors without pulse data were being classified, the pulse features were excluded from the vectors, but the participants from the supervised sessions were included.

The classification algorithms, KNN and SVM, were used as implemented in Scikit Learn by Buitinck et al. [2013]. Each method was tested with different hyperparameters against the training dataset using k-fold cross-validation, and later verified against the previously unseen validation set.

After the k-fold cross-validation was complete using the training examples, the classifier with the highest score was tested with the validation set. Thus leading to a precision, recall and kappa score for the classifier.

K-nearest neighbours The KNN algorithm was tested with three different hyperparameters: different values for K, either the distance or uniform weights and either the value 1 or 2 for p, making the distance metric either Manhattan or Euclidean respectively. The values for K were chosen based on the class limit set by the researcher. The value for K was all integers from 3 and up to half the class limit minus 1, but never greater than 8.

Support vector machine As for the SVM algorithm, the kernels RBF and linear were used as hyperparameters. For the RBF kernel, gamma values of 1, 0.1, 0.01, 0.001 and 0.0001, and C values of 1, 10, 30 and 50 were tested. The linear kernel was tested with values 1, 10, 30 and 50 for C. These values were selected as arbitrary values, found fitting for the dataset.

5.3.3 Experiment plan

KNN and SVM were experimented in a range of different approaches. It has been attempted to both classify emotional states universally, as done by Epp [2010], and individually, as done by Kołakowska [2015].

Universal classifications means that data samples from all users have been aggregated together into one dataset with six discrete classes. With the individual classifications, only data samples from one specific user were considered at a time. Due to the small number of data samples from each participant, and the large variation in the number of data sample per class collected, only a subset of the participants and classes were explored in some cases.

Additionally, both multiclass and binary classifiers were tested. In a multiclass classifier, there are more than two classes. A binary classifier consist of only two classes. In those cases, all classes except one were aggregated into one class, called “other”, which was classified against the remaining class.

With the universal classifier, a number of approaches were experimented with:

1. Multiclass classifier with all six emotional classes.

2. Multiclass classifier with all six emotional classes, but only using instances where the pulse has been measured (i.e. the pulse feature was greater than zero).
3. Multiclass classifier only including the four emotional classes with the most instances: Bored, concentrated, delighted, frustrated.
4. Multiclass classifier only including the four emotional classes with the most instances: Bored, concentrated, delighted, frustrated. But only using instances where the pulse has been measured (i.e. the pulse feature was greater than zero).
5. Binary classifier where every emotional class was classified against the five others, e.g. bored was classified against the class “other”, which was aggregated from concentrated, confused, delighted, frustrated and surprised.
6. Binary classifier where every emotional class is classified against the five others, e.g. bored was classified against the class “other”, which was aggregated from concentrated, confused, delighted, frustrated and surprised. But only using instances where the pulse has been measured (i.e. the pulse feature was greater than zero).

Experiments done with the pulse feature also include a comparison with a classifier where the same feature vectors were used, but did not have the pulse features.

With the individual classifiers, the approaches experimented with were:

1. Multiclass classifier with all emotional classes available for that participant
2. Binary classifier with all emotional classes available for that participant

For those participants who took part in the supervised data collections, where their pulse was measured, the experiments include both feature vectors with and without the pulse features.

As mentioned, to classify a sequence of keystrokes, the sequences must have identical characters, in the same order. These sequences were found by recognising a pattern in the collected data. Patterns used to detect keystroke sequences were inspired by the book *Eloquent JavaScript* by Haverbeke [2014]. In that book keywords and reserved words from JavaScript are listed. Programming language specific keywords were chosen because knowing the programming language of choice is an important skill when mastering programming. This in part means to know the most common keywords. Patterns were additionally chosen based on JavaScript 2015 keywords not listed in the book (e.g. “const” and “let”), and

phrases the participants were asked to type while doing Adapt’s exercises (e.g. “complete this tutorial”).

The most frequent pattern found was “console.log(“ (the opening parenthesis is intentional) as each exercise asked the participant to print results to the console. From the dataset 530 instances of this pattern was found. The pattern with second most data samples was “let” with 43 instances.

“console.log” was also tested as a pattern, which had 558 data samples before removing the outliers. However, preliminary results showed that choosing “console.log” as a pattern had a slightly less correct classification rate, thus there were not any gain from choosing a shorter pattern. Additionally, it has been hypothesised by Epp [2010] that special keys (e.g. numbers, punctuation marks and shift key) might be an important feature, thus the shift and left parenthesis keys were included in the pattern. Every result presented, both for the universal and individual classifiers, has been found using the pattern “console.log(“.

5.4 Population overview

In this section an overview of the participants making up the population for this study are presented. All data on the participants is anonymised and can’t be linked to one specific person. In table 5.3, an overview of each participant can be seen, ordered by how many sessions they generated (a session can be one character short, or code for an entire exercise).

In total 23 people participated in the study, of which 9 were females and 14 were males. Two participants did not have any previous experience with programming, while two reported to be master students (3.5 years or more of programming experience after first attending the university).

The participants reported to have been practicing programming an average of 1.24 years (0.95 when the master students are removed from the population), with a population standard deviation (SD) of 1.12 (0.65 without the master students). The reported average experience level was 3.04 (2.86 without the master students), with a SD of 1.16 (1.04 without the master students). This means that most participants had fairly little previous experience with programming. As such most participants fell within the targeted population.

10 participants had previous experience with JavaScript. The questionnaire did not ask participants to specify how well they knew certain languages, making it difficult to assess if they knew JavaScript well, or just barely. 20 participants had previous experience with Java, and 20 with Python. These two subsets were not entirely identical. Of the other languages, 3 reported to know C++, 3 knew Matlab and 1 new PHP. Additionally, some reported to know languages not specified, these were: C (1), C# (1), HTML (2), CSS (2), Rust (1) and ActionScript (1). The number of participants that reported the language is denoted in the

Table 5.3: Information about the participants, order by sessions

ID: Participant ID from table 6.8, Gen: gender, Exp: experience,
 FL: first learned programming, YBU: years before attending university they started
 learning programming, YOU: total years of programming experience, JS: JavaScript,
 JA: Java, PY: Python, MA: Matlab, #: Total number of sessions

ID	Gen	Exp	FL	YBU	YOU	JS	JA	PY	C++	MA	PHP	#
C	M	5	U		4.5	x	x	x				100
A	F	4	H	1	2.0		x			x		92
F	F	2	U		1	x	x	x				86
B	M	3	H	0.5	1.0		x	x	x			75
D	F	3	U		1.5		x	x				62
H	M	3	U		2.0			x	x	x		62
E	F	2	U		0.5	x	x	x				60
	M	3	H	1	2.0	x	x	x	x		x	48
	M	3	U		1.0		x	x				47
	M	4	U		1.0		x	x				46
	F	4	U		1.0	x	x	x				45
G	M	3	U		0.5	x	x	x				42
	F	2	U		0.5		x	x				37
	M	5	S	1	2.0	x	x	x				36
	M	3	U		0.5		x	x		x		36
	M	1	S									24
	M	2	U				x	x				18
	F	4	S	0.5	1.5		x	x				14
	F	4	H		1.0	x	x	x				11
	M	1	F									11
	M	5	U		4.0	x	x	x				9
	F	2	U		0.5	x	x	x				1
	M	2	U		0.5		x	x				1

parentheses. Although HTML and CSS are not a programming language, just a markup and styling language respectively, they are often used together with JavaScript and is worth mentioning in a programming language context.

Chapter 6

Results

The results from the experiments described in chapter 5 are presented in this chapter.

The results are aggregated into tables. The hyperparameters for each unique classifier are presented below the table. In each experiment, precision, recall and kappa scores are calculated. Although the F1 score also was calculated by the classification library Scikit Learn (Pedregosa et al. [2011]), this score is not presented in the the results. This decision was made in order to limit the amount of information given, so that the reader is not overwhelmed and that important measurements are not drowned in other information.

6.1 Universal classifications

In this section, six different experiments is presented, all done with both KNN and SVM classifiers and different sets of hyperparameters:

- Universal multiclass classifier
- Universal multiclass classifier with pulse
- Universal subset multiclass classifier
- Universal subset multiclass classifier with pulse
- Universal binary classifier
- Universal binary classifier with pulse

There are a total of 521 data samples for the pattern “console.log(“ after removing outliers. The class distribution for the dataset, both the entire dataset and the dataset that only includes instances with pulse features, can be seen in table 6.1. As the class surprised has 0 instances in the pulse dataset, it was excluded from the pulse experiments.

Table 6.1: Class distribution

BO: bored, EC: concentrated, CO: confused,
DE: delighted, FR: frustrated, SU: surprised

Dataset	BO	EC	CO	DE	FR	SU
Without pulse	60	242	22	106	83	7
With pulse	12	116	4	11	14	0

6.1.1 Universal multiclass classifier

The results from the universal multiclass classifier can be seen in table 6.2. Due to undersampling, the number of instances per class was limited to 60, resulting in the following class distribution: Bored: 60, concentrated: 60, confused: 22, delighted: 60, frustrated: 60, surprised: 7. Precision and recall is noted as percentage, while the kappa score is a number between -1 and 1.

The KNN algorithm was tested with values 3, 4, 5, 6, 7 and 8 as values for K. All hyperparameter combinations and how they compare to the other combinations can be seen in figure 6.1. The box is the average precision score, and the line in the center of each box is the difference between the maximum score and the minimum score for that classifier. The best hyperparameters found was weights: distance, p: 1, and k: 3 (blue bar, second from the left, with an average precision score of 0.292 +/- 0.10.5), which resulted in an average precision of 34% and an average recall of 37%, with a kappa score of 0.19. The classes were classified as follows in order of precision (precision and recall are written in parentheses): Bored (45%, 42%), delighted (44%, 63%), concentrated (39%, 37%), frustrated (18%, 20%), and last confused and surprised (0%, 0%).

The different hyperparameter combinations for the SVM classifiers can be seen in figure 6.2. The best hyperparameters for the SVM algorithm were discovered to be: kernel: RBF, C: 30, gamma: 1 (as can be seen in the blue bar, the 11th bar from the left, with average precision score of 0.390 +/- 0.154), which resulted in an average precision of 39% and an average recall of 42%, with a kappa score of 0.25. The classes were classified as follows in order of precision (precision and recall are written in parentheses): Delighted (59%, 53%), concentrated (50%, 37%), bored (36%, 74%), frustrated (27%, 20%), and last confused and surprised (0%, 0%).

Table 6.2: Results from the universal multiclass classifier

Emotion	Method	Precision	Recall
Bored	KNN ¹	47	42
	SVM ²	36	74
Concentrated	KNN	39	38
	SVM	50	37
Confused	KNN	0	0
	SVM	0	0
Delighted	KNN	44	52
	SVM	59	53
Frustrated	KNN	18	52
	SVM	27	20
Surprised	KNN	0	0
	SVM	0	00
Avg / total	KNN	34	37
	SVM	39	42
Kappa score	KNN	0.19	
	SVM	0.25	

¹Weights: distance, P: 1, k: 3 ²Kernel: RBF,
C: 30, gamma: 1

SVM generally slightly outperformed KNN, both on precision and recall, except for the class bored.

6.1.2 Universal multiclass classifier with pulse

In this experiment, only feature vectors where the pulse feature was greater than zero was included. The results are presented in table 6.3. As a comparison, the same feature vectors are classified with the pulse features excluded. In the instances where the precision or recall is equal for both vectors, i.e. with pulse included and excluded, the table cells have been merged to better visualise the values being the same. The classes were limited to a maximum of 15 instances, resulting in the following class distribution: Bored: 12, concentrated: 15, confused: 4, delighted: 11, frustrated: 14.

The KNN algorithm was tested with values 3, 4, 5, 6, and 7 as values for K. The best hyperparameters found for the classifier was weights: distance, p: 1, and k: 5. For the classification with the pulse features included, this resulted in an average precision of 51% and an average recall of 47%, with a kappa score of 0.26. For the classification with the pulse features excluded, this resulted in

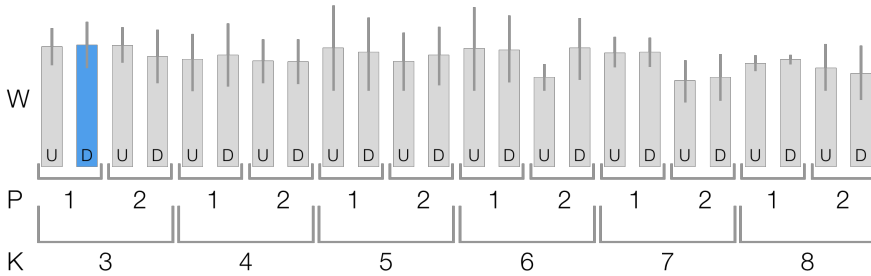


Figure 6.1: Comparison of all KNN classifier combinations for the universal multiclass classifier.

an average precision of 54% and an average recall of 47%, with a kappa score of 0.29.

The best hyperparameters for the SVM algorithm was found to be: kernel: linear, C: 1. This resulted in an average precision of 51% and an average recall of 41%, with a kappa score of 0.17 for both classifiers.

None of the universal classifiers with pulse could classify bored and confused, probably due to the small number of instances. Including the pulse features led to a worse classification than when excluding it. KNN generally slightly outperformed SVM, both when the pulse features were included and excluded, except for the class frustrated.

6.1.3 Universal subset multiclass classifier

In this experiment, the two classes with the fewest instances (confused and surprised) were excluded. The results from the universal multiclass classifier performed on the remaining classes can be seen in table 6.4. The number of instances per class was limited to 120, resulting in the following class distribution: Bored: 60, concentrated: 120, delighted: 106, frustrated: 84. This increase in data samples was done to get larger training and validation sets, and because the class skew was not that prominent with confused and surprised excluded.

The KNN algorithm was tested with values 3, 4, 5, 6, 7 and 8 as values for K. The best hyperparameters found was weights: uniform, p: 1, and k: 3, which resulted in an average precision of 48% and an average recall of 42%, with a kappa score of 0.23. The classes were classified as follows in order of precision (precision and recall are written in parentheses): Delighted (69%, 43%), concentrated (41%, 56%), frustrated (35%, 28%) and bored (25%, 41%).

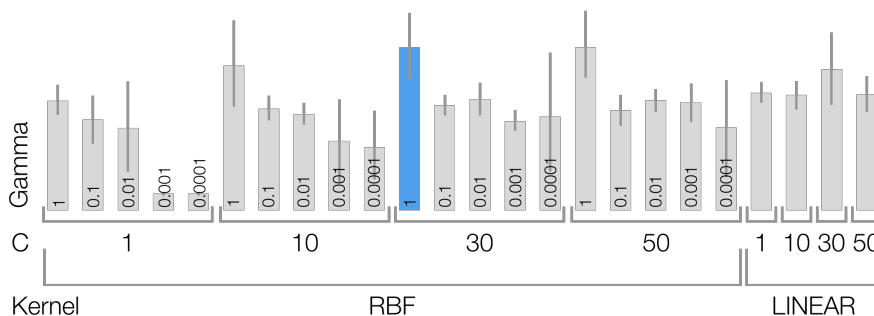


Figure 6.2: Comparison of all SVM classifier combinations for the universal multiclass classifier.

The best hyperparameters for the SVM algorithm were found to be: kernel: linear, C: 30, which resulted in an average precision of 53% and an average recall of 52%, with a kappa score of 0.33. The classes were classified as follows in order of precision (precision and recall are written in parentheses): Delighted (61%, 64%), bored (57%, 47%), frustrated (52%, 44%) and concentrated (38%, 44%).

SVM somewhat outperformed KNN on classes bored and frustrated, while KNN slightly outperformed SVM on classes concentrated and delighted.

6.1.4 Universal subset multiclass classifier with pulse

In this experiment, the two classes with the fewest instances (confused and surprised) have been excluded, and only feature vectors where the pulse feature was greater than zero have been included. The results are presented in table 6.5. As a comparison, the same feature vectors are classified with the pulse features excluded. In the instances when the precision or recall are equal for both vectors, i.e. with pulse included and excluded, the table cells have been merged to better visualise that the values are the same. The classes were limited to a maximum of 15 instances, resulting in the following class distribution: Bored: 12, concentrated: 15, delighted: 11, frustrated: 14.

The KNN algorithm was tested with values 3, 4, 5, 6, and 7 as values for K. The best hyperparameters found for the classifier were weights: distance, p: 1, and k: 6. For the classification with pulse features included, this resulted in an average precision of 35%, an average recall of 44% and with a kappa score of 0.17. For the classification with pulse features excluded, this resulted in an average precision of 42% and an average recall of 50%, with a kappa score of 0.28.

The best hyperparameters for the SVM algorithm was found to be: kernel:

Table 6.3: Results from the universal multiclass classifier using pulse

Class / Total score	Method	Precision (pulse)	Precision	Recall (pulse)	Recall
Bored	KNN ¹	0		0	
	SVM ²	0		0	
Concentrated	KNN	50	56	83	67
	SVM	45		83	
Confused	KNN	0		0	67
	SVM	0		0	
Delighted	KNN	100		0.40	33
	SVM	100		20	33
Frustrated	KNN	33	40	50	57
	SVM	50		50	
Avg / total	KNN	51	54	47	
	SVM	51		41	
Kappa score	KNN	0.26	0.29	0.26	0.29
	SVM	0.17		0.17	

¹ Weights: distance, P: 1, K: 5 ² Kernel: linear, C: 1

RBF, C: 30, gamma: 0.1. For the classification with pulse features included, this resulted in an average precision of 34% and an average recall of 38%, with a kappa score of 0.14. For the classification with pulse features excluded, this resulted in an average precision of 48% and an average recall of 44%, with a kappa score of 0.24.

Again, including pulse features led to a worse classification than when excluding it. SVM somewhat outperformed KNN on classes bored and frustrated, but it failed to classify any instances of delighted. KNN somewhat outperformed SVM on classes delighted, but failed to classify any instances of bored. With the class concentrated both SVM and KNN performed similarly.

6.1.5 Universal binary classifier

In this experiment, binary classifiers were trained and tested. Five classes were aggregated into a new class called “other”, while the sixth class remained. The remaining class was then classified against the class “other”. This was done for all six classes. The class limit was set equal to the instances of one class, i.e. for bored the class limit was 60, while for confused it was 22. This was done to have the equal amount of instances for each class, but also to maximise the number of classes wherever possible. The results can be seen in table 6.6. Two things should

Table 6.4: Results from the universal subset multiclass classifier

Emotion class / Total score	Method	Precision	Recall
Bored	KNN ¹	25	41
	SVM ²	57	47
Concentrated	KNN	41	56
	SVM	38	44
Delighted	KNN	69	43
	SVM	61	64
Frustrated	KNN	35	28
	SVM	52	44
Avg / total	KNN	48	42
	SVM	53	52
Kappa score	KNN	0.23	
	SVM	0.33	

¹ Weights: uniform, P: 1, K: 3 ² Kernel: linear, C: 30

be noted about the table: Parentheses behind the class name denotes the class limits, and the precision measure is the average for that classifier, following the example set by Kolakowska [2015].

Each classifier was trained and tested separately, thus the hyperparameters vary. Four out of six KNN algorithms used the uniform distance metric, while the two other used the distance metrics. The value for p was 1 in five of the classifiers, except for the sixth (frustrated). The value for K varied from 3 to 8. For the SVM classifiers, four of them used the RBF kernel, while two used the linear kernel. For those using the linear kernel, the value for C was 10 and 30. The algorithms using the RBF kernel had values between 10 and 50 for C, and gamma between 0.0001 and 1. The specific hyperparameters for each classifier can be seen below table 6.6.

Bored, confused and frustrated are the classes with the highest accuracy score, with scores of 75%, 71 and 74% respectively for KNN, and 71%, 79 and 85% respectively for SVM. The kappa score for KNN ranges from 0.43 to 0.45, and for SVM it ranges from 0.33 to 0.65. This can be considered a moderately good agreement.

The class surprised also had a high precision score, 80% for KNN and 100% for SVM. However, this class has a limited instances per class, with the validation set containing 3 and 2 instances of surprised and “other” respectively. The training set consisted of 4 and 5 instances, respectively. The SVM classifier correctly

Table 6.5: Results from the universal subset multiclass classifier with pulse

Emotion class / Total score	Method	Precision (pulse)	Precision	Recall (pulse)	Recall
Bored	KNN ¹	0		0	
	SVM ²	17	25	33	67
Concentrated	KNN	50	56	0.83	
	SVM	57	67	67	
Delighted	KNN	67	67	50	57
	SVM	0		0	
Frustrated	KNN	0	25	0	33
	SVM	50	100	33	
Avg / total	KNN	35	42	44	50
	SVM	34	48	38	44
Kappa score	KNN	0.18	0.28	0.18	0.28
	SVM	0.14	0.24	0.14	0.24

¹ P: 1, k: 6, weight: distance ² Kernel: RBF, C: 30, gamma: 0.1

classified all instances of surprised and “other” correctly. On the other hand, the KNN classifier only found one out of three instances of surprised, and classified all other instances as “other”.

KNN slightly outperformed SVM on the class bored, and significantly with the class concentrated. SVM slightly outperformed KNN on the classes confused, delighted and frustrated, and significantly on surprised.

Kołakowska [2015] created a similar universal binary classifier as the one presented in this study. The two classes present in both studies are bored and surprised. That study found a 58% precision for bored, and 53% precision for surprised, using KNN. While in this study, the precisions are 75% and 80%, respectively. However, the number of instances differ. In this study, the number of instances per class was 60 and 7, while in hers study the number instances per class was 45 and 19, respectively.

6.1.6 Universal binary classifier with pulse

In this experiment the classes confused and surprised have been excluded due to the small number of instances, 4 and 0 respectively. For the resulting four classes, three were aggregated into a new class called “other”, while the fourth class remained its original class. The remaining class was then classified against the class “other”. This was done for all four classes. The class limit was set equal to the number of instances for each class. The class limit is denoted in the

Table 6.6: Results from the universal binary classifier

Emotion class	Method	Precision	Recall for selected emotion	Recall for other emotions	Kappa score
Bored (60)	KNN ¹	75	56	89	0.44
	SVM ²	71	44	89	0.33
Concentrated (242)	KNN ³	60	66	54	0.20
	SVM ⁴	24	0	100	0.00
Confused (22)	KNN ⁵	71	71	71	0.43
	SVM ⁶	79	86	71	0.57
Delighted (106)	KNN ⁷	57	82	29	0.11
	SVM ⁸	68	85	45	0.30
Frustrated (84)	KNN ⁹	74	60	85	0.45
	SVM ¹⁰	85	68	96	0.65
Surprised (7)	KNN ¹¹	80	33	100	0.29
	SVM ¹²	100	100	100	1.00

¹ Weights: distance, p: 1, k: 4

² Kernel: linear, C: 10

³ Weights: uniform, p: 1, k: 8

⁴ Kernel: RBF, C: 10, gamma: 0.0001

⁵ Weights: uniform, p: 1, k: 8

⁶ Kernel: RBF, C: 30, gamma: 0.1

⁷ Weights: uniform, p: 1, k: 6

⁸ Kernel: linear, C: 30

⁹ Weights: distance, p: 1, k: 6

¹⁰ Kernel: RBF, C: 50, gamma: 0.01

¹¹ Weights: uniform, p: 2, k: 3

¹² Kernel: RBF, C: 1, gamma: 1

parentheses behind the class name. The results can be seen in table 6.7. This approach is similar to the one presented in section 6.1.5. As with the previous section, the precision measure is the average for that classifier, following the example set by Kolałowska [2015].

Each classifier was trained and tested separately, thus the hyperparameters vary. One of the four KNN classifiers used the uniform weights, while the three others used distance. The value for p was 2 with the first classifier, and 1 with the other three. The value for K was 3 in two of the classifiers, 8 in another, and 4 in the last. Three out of four SVM classifiers used the RBF kernel with the value for C being 10, 1 and 10, and the value for gamma being 1, 1 and 0.01. The fourth classifier used the linear kernel with 10 as the value for C. The specific hyperparameters for each classifier can be seen below table 6.7.

The first thing to note with the results in this experiment is that every score is equal between both vectors, with and without pulse features, i.e. adding the pulse features apparently does not give any additional information about the class.

Delighted and frustrated are the two classes that have the highest scores, with

Table 6.7: Results for the universal binary classifier with pulse

P = precision, (p) = pulse, KS = kappa score

Emotion class	Method	P (P)	P	Recall for selected (pulse)	Recall for selected	Recall other (pulse)	Recall for other emotions	KS (P)	KS
Bored (12)	KNN ¹	75		67		80		0.47	
	SVM ²	57		67		40		0.06	
Concentrated (37)	KNN ³	78		100		25		0.24	
	SVM ⁴	48		49		50		-0.05	
Delighted (11)	KNN ⁵	100		100		100		1.00	
	SVM ⁶	089		100		075		0.72	
Frustrated (14)	KNN ⁷	100		100		100		1.00	
	SVM ⁸	91		75		100		0.77	

¹ Weights: distance, p: 2, k: 3² Kernel: RBF, C: 10, gamma: 1³ Weights: distance, p: 1, k: 8⁴ Kernel: RBF, C: 1, gamma: 1⁵ Weights: uniform, p: 1, k: 3⁶ Kernel: RBF, C: 10, gamma: 0.01⁷ Weights: distance, p: 1, k: 4⁸ Kernel: linear, C: 10

100% precision and recall in KNN. In SVM, the same classes have a precision of 89% and 91% respectively. Both classes have a high kappa score with all four classifiers.

Concentrated has a fairly high precision score with the KNN classifier. The recall for concentrated is also very good, but a high number of “other” instances have also been classified as concentrated. The SVM classifier has a moderate precision and recall score, but the kappa score is negative, meaning that the results are less than could be attributed to chance alone.

In this experiment, KNN outperformed SVM on all classes, and SVM performed terrible on the classes bored and concentrated. As there are no differences between the vectors with and without pulse features, it is possible to hypothesize that the pulse feature carry no information gain for the classification. However, it is important to remember that the pulse features account for three out of 156 features in the vector, thus their influence is minimal.

6.2 Individual classifications

In this section, the two individual experiments are presented, all done with both KNN and SVM classifiers and different sets of hyperparameters:

- Individual multiclass classifier
- Individual binary classifier

The 8 participants with the highest number of sessions have been chosen, including those attending the supervised data collection sessions. The class distribution for each participant can be seen in table 6.8. The participants with a p in parentheses (D through H) are the ones that had their pulse measured. Their results are shown with both vectors classified, with and without the pulse feature. Registered information on these participants can be seen in table 5.3.

Table 6.8: Individual class distribution

BO: bored, EC: concentrated, CO: confused, DE: delighted,
FR: frustrated: #: total number of instances

Participant	BO	EC	CO	DE	FR	SU	#
A	4	21	9	4	52	2	92
B	1	39	3	4	28	0	75
C	0	7	15	63	15	0	100
D (p)	0	37	0	0	25	0	62
E (p)	3	45	1	2	9	0	60
F (p)	4	39	5	18	19	1	86
G (p)	2	33	4	1	1	1	42
H (p)	5	34	5	4	14	0	62

Classifications resulting in 0 correct classifications are excluded from the results. Additionally, those resulting in a kappa score of 0 or less are also excluded, as these can be considered random or less than random. In the individual binary classifier, classes with too few instances to be classified are also excluded from the results.

Table 6.9: Information about individual participants

Participants selected for individual experiments,
the information is self reported.

Participant	Level of experience	Knows JavaScript	Years of experience
A	4		2
B	3		1
C	5	Yes	4.5
D	3		1.5
E	2	Yes	0.5
F	2	Yes	1
G	3	Yes	0.5
H	3		2

Information about the selected participants can be seen in table 6.9. All participants have taken the introductory programming course at NTNU, either

learning Python or Matlab. The user registration form did not ask to what degree the participants knew a certain programming language. Thus it is not possible to assess how proficient they are in JavaScript. Participant C is a student at the masters level, and differ thus from the other participants presented here.

In the multiclass experiments, the results from participants A, B, C, D, and H yielded for some scenarios a classification better than random. With the binary experiments, participants A, B, C, D, F and H got positive results from some emotional states. Concentrated was somewhat correctly classified for four participants, frustrated was also somewhat correctly classified for four participants, while it was possible to somewhat correctly classify confused for only one participant. No other emotional states were classified correctly with a kappa score greater than zero.

No classifier managed to classify any of participants E's and G's emotional states; This was the case in both the multiclass and binary experiments.

6.2.1 Individual multiclass classifier

In this experiment, an individual multiclass classifier was trained and verified for each participant. The number of instances per class was limited for each participant to create a more uniform class distribution. The class limit was a compromise between creating a uniform class distribution and using as many instances as possible.

The results for participants A, B, C and H can be seen in tables 6.10, 6.11, 6.12 and 6.13, respectively. Class distribution is denoted in the parentheses after the class name. The classes that had a precision and recall score of zero for both KNN and SVM classifiers have been excluded from the table, but are mentioned in the table's description. Meaning that the instances were classified, but are not presented in the table. Class distribution is denoted in the parentheses behind the class name.

Some classifiers have "n/a" denoted instead of the scores, this means that the classifier did not output a classification score for that class.

For participant A (see table 6.10), even though the precision and recall score have a moderate score, the number of instances available in the validation set is small. Thus the classification is close to random for KNN, and equal to random for SVM, as can be seen from the kappa score.

Participant B's KNN classifier, see table 6.11, however has a kappa score of 0.44, which is decent. Although some of the other classes were classified falsely, all concentrated and frustrated instances were classified correctly. This was also the case with participant C's KNN classifier, see table 6.12, except that the classifier failed to classify any of the five instances of frustrated correctly. Participant C's SVM classifier had a negative kappa score, making the classification less precise

Table 6.10: Individual multiclass classifier for participant A

Class limit: 10. Delighted (3) and frustrated (4)
had a score of 0 for precision and recall.

Emotion class	Method	Precision	Recall
Bored (4)	KNN ¹	00	50
	SVM ²	n/a	
Concentrated (8)	KNN	50	33
	SVM	50	100
Avg / total	KNN	58	33
	SVM	25	50
Kappa score	KNN	0.14	
	SVM	0.00	

¹ Weight: uniform, p: 2, k: 3

² Kernel: RBF, C: 1, gamma: 1

Table 6.11: Individual multiclass classifier for participant B

Class limit: 10. Confused (3) and delighted (2)
had a score of 0 for precision and recall.

Emotion class	Method	Precision	Recall
Concentrated (10)	KNN ¹	60	100
	SVM ²	0	
Frustrated (10)	KNN	67	100
	SVM	40	100
Avg / total	KNN	39	62
	SVM	10	25
Kappa score	KNN	0.44	
	SVM	0.11	

¹ Weight: uniform, p: 1, k: 3 ² Kernel: linear, C: 10

than by chance alone.

In table 6.13 both scores for pulse and non-pulse feature vectors are denoted for participant H. As can be seen, there is no difference between the classifiers when the pulse features are present and when they are not. This participant has a generally low recall rate, except for frustrated with the KNN classifier, indicating that many of the other instances also was classified as frustrated. The kappa score for both classifiers are close to random, SVM being less than random.

6.2.2 Individual binary classifier

Similar to the binary classifiers presented in sections 6.1.5 and 6.1.6, all classes except one have been aggregated into a new class “other”, and then the remaining

Table 6.12: Individual multiclass classifier for participant C

Class limit: 15. Frustrated (15) had a score of 0 for precision and recall.

Emotion class	Method	Precision	Recall
Concentrated (7)	KNN ¹	67	100
	SVM ²	0	
Confused (8)	KNN	50	
	SVM	50	
Delighted (15)	KNN	75	43
	SVM	50	14
Avg / total	KNN	54	43
	SVM	32	14
Kappa score	KNN	0.22	
	SVM	-0.13	

¹ Weight: distance, p: 1, k: 6

² Kernel: RBF, C: 1, gamma: 1

class has been classified against that class. This has been done for all classes, and for each selected participant. The class limit is denoted after the class name in each table. This limit is set to the number of instances for that class, generating as large datasets as possible.

Classifiers that failed to classify any instances correctly, or that have a kappa score of zero or less, have been excluded from the tables. The exception is when one of the classifiers managed to get a promising classification, then the other classifier is included as a comparison.

Participant A's KNN classifier, see table 6.14, had a fair classification score, and a satisfactory kappa score. However, it should be noted that the validation set consisted of 7 instances for each class, as such, one correct classification means an apparent high increase in accuracy. The kappa score of 0.67 gives this classifier a good merit.

Participant C's SVM classifier for frustrated, see table 6.16, resulted in a moderately high classification rate, correctly classifying all instances of frustrated correctly, although also classifying two instance of the class "other" as frustrated. For the KNN confused classifier, it should be noted that the validation set for the class confused only consisted of one instance, resulting in a high overall recall and precision. But it falsely classified instances of the class "other" as confused.

Classifiers for participants B, D, F and H, see tables 6.15, 6.17, 6.18, and 6.19 respectively, however have a low kappa score, indicating that these classifications would be equal to a random classification.

For participant H both vectors, with and without the pulse features, resulted in the same score. This is almost also the case with participant F, however here

Table 6.13: Individual multiclass classifier for participant H

Class limit: 10. Bored (4) and delighted (3)
had a score of 0 for precision and recall

Emotion class	Method	Precision (pulse)	Precision	Recall (pulse)	Recall
Concentrated (10)	KNN ¹	1.00		0.20	
	SVM ²	67		40	
Frustrated (7)	KNN	20		100	
	SVM	0		0	
Avg / total	KNN	65		25	
	SVM	42		25	
Kappa score	KNN		0.08		
	SVM		-0.12		

¹ Weights: distance, p: 2, k: 5 ² Kernel: RBF, C: 50, gamma: 0.01

there are small differences in the scores.

Table 6.14: Individual binary classifier for participant A

Emotion class	Method	Precision	Recall for selected emotion	Recall for other emotions	Kappa score
Concentrate (21)	KNN ¹	88	67	100	0.67
	SVM ²	50	33	67	0

¹ Weights: uniform, p: 1, k: 5 ² Kernel: RBF, C: 10, gamma: 0.01

Table 6.15: Individual binary classifier for participant B

Emotion class	Method	Precision	Recall for selected emotion	Recall for other emotions	Kappa score
Concentrate (39)	KNN ¹	50	17	83	0.00
	SVM ²	59	50	67	0.17
Frustrated (28)	KNN ³	57	80	29	0.08
	SVM ⁴	67	40	86	0.27

¹ Weights: uniform, p: 1, K: 5 ² Kernel: linear, C: 10

³ Weights: uniform, p: 1, k: 4 ⁴ Kernel: RBF, C: 1, gamma: 1

Table 6.16: Individual binary classifier for participant C

Emotion class	Method	Precision	Recall for selected emotion	Recall for other emotions	Kappa score
Confused (15)	KNN ¹	90	100	67	0.36
	SVM ²	71	0	83	-0.17
Frustrated (28)	KNN ³	59	40	75	0.14
	SVM ⁴	84	100	50	0.53

¹ Weights: distance, p: 2, k: 5 ² Kernel: RBF, C: 10, gamma: 1

³ Weights: uniform, p: 1, k: 4 ⁴ Kernel: RBF, C: 30, gamma: 1

Table 6.17: Individual binary classifier for participant D

P: precision, (p): pulse, RS: recall for selected emotion,

RO: recall for other emotions, KS = kappa score

Emotion class	Method	P (P)	P	RS (P)	RS	RO (P)	RO	KS (P)	KS
Frustrated (25)	KNN ¹	64		0		100		0.00	
	SVM ²	90	68	100	50	75	50	0.55	0.00

¹ Weights: uniform, p: 1, K: 5 ² Kernel: linear, C: 30

Table 6.18: Individual binary classifier for participant F

P: precision, (p): pulse, RS: recall for selected emotion,

RO: recall for other emotions, KS = kappa score

Emotion class	Method	P (P)	P	RS (P)	RS	RO (P)	RO	KS (P)	KS
Concentrated (39)	KNN ¹	56	84	100		0	8	0.00	0.04
	SVM ²	83	70	33	33	100	75	0.20	0.07

¹ Weights: uniform, p: 1, k: 8 ² Kernel: RBF, C: 10, gamma: 1

Table 6.19: Individual binary classifier for participant H

P: precision, (p): pulse, RS: recall for selected emotion,
 RO: recall for other emotions, KS = kappa score

Emotion class	Method	P (P)	P	RS (P)	RS	RO (P)	RO	KS (P)	KS
Concentrated (17)	KNN ¹	43		33		50		-0.15	
	SVM ²	57		83		25		0.09	
Frustrated (14)	KNN ³	90		100		67		0.36	
	SVM ⁴	89		100		50		0.22	

¹Weights: uniform, p: 2, k: 7 ²Kernel: RBF, C: 1, gamma: 1

³Weights: uniform, p: 1, k: 6 ⁴Kernel: RBF, C: 10, gamma: 1

Chapter 7

Evaluation, contributions and future work

In this chapter, the results presented in chapter 6 will be discussed and evaluated. Moreover, the contributions to the research field will be presented. And lastly, some propositions for the way forward will be made.

Firstly however, a summary of the research questions are necessary. The overall goal for this research was to see if it was possible to detect a learner's emotional state non-intrusively in a programming tutorial. This research does not give a definitive answer to this, however, it is one of many steps that has been, and must be, taken for the answer to be a single yes.

The goal was divided into two subquestions:

Research question 1 *Is it possible to detect a learner's emotional state using keystroke dynamics from programming keywords?*

Research question 2 *Will detecting a learner's pulse yield a better classification than typing rhythm alone?*

With these questions in mind, the results can be discussed and the research evaluated.

7.1 Evaluation and discussion

In this section, the data collection and experiment phases will be evaluated before the research questions are answered as far as this research's results allow.

7.1.1 Data collection phase

During the course of the data collection phase, Adapt received four new features (user information form, emotion inducing features, disabling pasting code into the editor, and disabling undoing). This means that not all participants were tested with the same features. It also means that not all participants registered information about themselves, which led to the dataset having 269 fewer sessions than what was collected in total. However, it was important to the researcher that every user could be categorised if needed, and that the dataset was consistent across experiments as far as necessary.

7.1.1.1 Participants

In this research, a population of 23 participants were used, where two contributed with as little as 1 session. Epp [2010] reported that 26 users took part in the study, but only the 12 with more responses than 50 were used during the classification. Choosing a similar approach would have left this study with only 7 participants. It is not clear how many responses in total were used in Epp's research. This research did generate a total of 963 sessions, but the quality of each session is varying as not all could be used during classification (the once that didn't contain the keyword "console.log("). This is more samples than the 207 from Kołakowska [2015] 9 users, however, it might happen that all those data samples could be used during classification.

It must be mentioned, however, that participants in this research most often did the tutorial once, while Epp [2010] and Kołakowska [2015] gathered data over a longer period of time.

7.1.1.2 Emotion inducing features

That the emotion inducing features (random text removal and timer) were not implemented earlier does not affect the dataset in the researcher's eyes, as sessions before the implementation would be equal to those where they did not occur.

The effect these emotion inducing features had on the participants needs some exploration to see if they did what was intended.

The timer feature occurred a total of 138 times. Out of those, participants reported that it had an effect in 102 of the instances. When it did not have any effect, participants wrote that it was because they either had plenty of time for the exercise, or that the timer already had reached zero (in which case it was still visible, but not counting). In the cases where the timer did have an effect, participants reported that it made them stressed, irritated, more focused, and made them more motivated to do the exercise quickly. It should be noted,

however, that the text answer was not required, as such not all reports had an explanation for how it affected the participants.

The random text removal occurred a total of 126 times, of which it was reported 97 times that it did affect the participants. When it did not have an effect, participants reported that they had gotten used to it, they were expecting it to happen, that it did not remove any important text, or that they could simply undo their last action and get the code back. The latter being the reason for disabling the undo function. When the random text removal had an effect, participants reported that it was irritating, annoying and frustrating, some felt that they had done something wrong, and one speculated what triggered the text deletion. However, participants got used to it after some time, expecting it to happen at any time. Also here some answers remained empty.

From this feedback it can be assumed that the timer and text deletion did have an effect as intended. The surprise effect of both features fades away after they have happened some times. Though, the text deletion was seen as very annoying on the last exercises as they demanded more code, and the chance to encounter it was greater.

7.1.1.3 Selecting emotional states

Discrete emotional classes were selected instead of coordinates in a two-dimensional space. Although the latter approach could have yielded more accurate emotions, participants would have also needed to determine how aroused they were, and how positive their emotion was. This could have lead to a more fragmented dataset, where the researcher had to decide which coordinates related to each emotion, possibly by using the model by Kort et al. [2001]. In order to make it easier for the participants, six discrete classes were chosen.

As noted by Kołakowska [2015], the researcher needs to trust that participants label their own sessions as correctly as possible. There is no guarantee that there does not exist any mislabeled sessions. This is a risk the researcher was willing to take to get as many data samples as possible. Additionally, the researcher is not an expert at emotions, and could thus not be a judge of which emotional states were observed.

More likely than deliberate mislabeling of data samples, however, are unconsciously mislabeling. During the observed data collections, some participants seemed unsure about their emotional state, and after selecting one of the pre-defined, selected another. This could mean that emotional states are somewhat overlapping, or that neither of them fitted their real emotional state, thus trying to find the closest one. This could lead to mislabeled data. One might handle this by having another checkbox which the participants can select if neither pre-defined emotional states represent their current state, and a free text field where they can describe their emotional state.

Another approach to dealing with this challenge is the way it was done by Epp [2010]. In that study, participants were asked to answer how present each of the 15 emotional states were, on a 5-point Likert scale. This, however, would have led to more classes, and possibly fewer instances of each class. Using six discrete classes was seen as the best approach to balance the number of classes and the number instances in each class.

It should also be mentioned that the researcher did not define the emotional states for the participants. Thus they had to use their own definition of the emotions, which may vary between participants.

7.1.1.4 Collecting participant's pulse

During the observed data collection sessions, the researcher wanted to capture both the participant's screen and the phone displaying the participant's pulse, as to sync the pulse to the correct session. However, this led the phone to be visible to participants. This allowed participants to see their own pulse, and possibly try to relax themselves when it got too high. During the third observed session, the phone was moved further away, making it harder for the participant to look at it. Participant D, which was the third to take part in the observed data collection, said after the session that she had focused the entire time to keep her pulse low, even though she did not watch the phone. Thus she was affected by the environment.

Participant H, which was the fifth and last to take part in the observed data collection, did the exercises over a course of two days because of limited time the first day. He started with a pulse of 68 the first day, and a pulse of 79 the second day. He mentioned that he did not sleep well the night before. These were factors that the researcher had not anticipated, and could lead to a noisy dataset.

The researcher also had no information about the participants' resting pulse. Using the resting pulse, it would have been possible to measure the variance per user, instead of assuming that the pulse measures meant the same for everyone. If the Fitbit activity tracker had been an integrated part of the system, and every participant had used one of their own, it could have been possible to gather such data. The researcher asked all of the participants if they knew their resting pulse, but none did.

The Fitbit also posed a challenge for participant D, as she had thin arms. At three different times, the Fitbit could not get a read of the pulse. Luckily this happened while the participant was not typing, but it could have been a problem if it occurred more often. It should be mentioned that for those instances for which there were no pulse data, the pulse data remained zero in the session object.

Additionally, the researchers presence affected at least one participant's performance, making her stressed. Although the researcher sat behind the partici-

pant, out of sight, and tried to keep quiet. There is no guarantee that the presence didn't also affect others, but only the two participants reported it. Before the data collection started, it was stated by the researcher that it wasn't the participants who were tested, and that they could choose any emotional state, even bored. It was not pointed out that the researcher had created the exercises.

All of these factors indicate that the supervised sessions need refinement, perhaps observing the participants while they are studying normally. Coming into a lab will always be unnatural.

7.1.1.5 Exercises

Exercises were written by the researcher, based on his knowledge of JavaScript and experiences with teaching this to first year informatic students. The exercises were proofread by one native English speaker and one first year student who had already attended the course IT2805 Web technologies. However, the researcher does not have any pedagogically education, thus there is no guarantee that the exercises were as good as they could have been. They were never checked by a lecturer at the department. The exercises followed loosely the same progress as done by Codeacademy, and in the course IT2805.

More importantly perhaps, the exercises were made to be easy to follow, and steadily increasing in difficulty, as would a normal tutorial. However, in the setting of capturing different emotional states, the exercises could have been created such that they provoked certain emotions.

In the beginning of the data collection, there were four exercises on the topic of arithmetic. These were merged together after feedback that it was boring doing almost the same thing four times. Only after the data collection phase was over did it dawn upon the researcher that this could have been used to provoke participants into becoming bored

As a way to provoke some emotions, however, the emotion inducing features were added, and two unexpected exercises with a timer were added in the middle of the tutorial.

7.1.2 Experiment phase

The most obvious limitation for these experiments was the limited dataset, as also reported by both Epp [2010] and Kołakowska [2015]. Longi et al. [2015] is the most closely related research that has reported to have a large dataset. Following those lines, it would be interesting to gather data through the programming exercises in IT2805 Web technologies, over the course of one entire semester.

During the experiments, only the pattern `console.log(` was investigated thoroughly, and not any subpatterns of `console.log` either. As such, it may happen

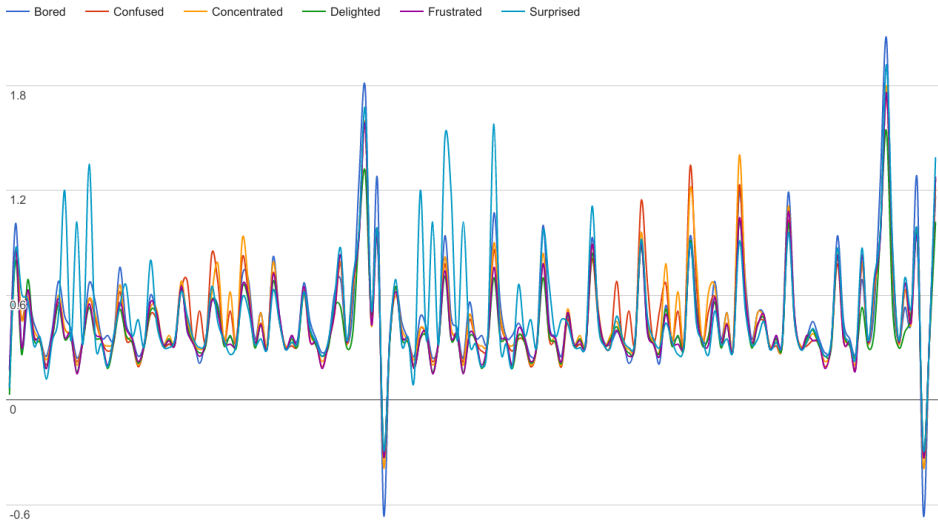


Figure 7.1: Comparison of average feature vectors for all six classes. Y-axis denotes the normalised values, x-axis denotes the features 1 through 153 (see appendix E for features and their corresponding values).

that part of the string contains more discriminative features for emotion recognition. No feature selection on the constructed feature vectors were done before classifying, which may have led to less accurate classifications than what could have been achieved. This is perhaps the biggest drawback of the classification process.

Additionally, when cleaning up the sequences, as described in section 5.3.2.1, keystrokes important to the classification might have been removed. Removing these keys also lead to an unnatural long pause between keys, which would result in noisy data. This could have been overcome by only classifying sequences without any mistakes. However, that would probably lead to an unreal dataset, as typing errors occur often. Additionally, backspace and delete keystrokes are seen as important features, and should be preserved in some way. Removing outliers did remove at least some of these unnatural sequences.

In figure 7.1, average feature vectors for all classes can be seen. These curves represent the pattern “console.log(“, and features are distributed along the x-axis, while their normalised value are represented by the y-axis. The class surprised differ the most from the rest at the beginning and in the middle, while the class concentrated differ most at the beginning of the last third. This graph

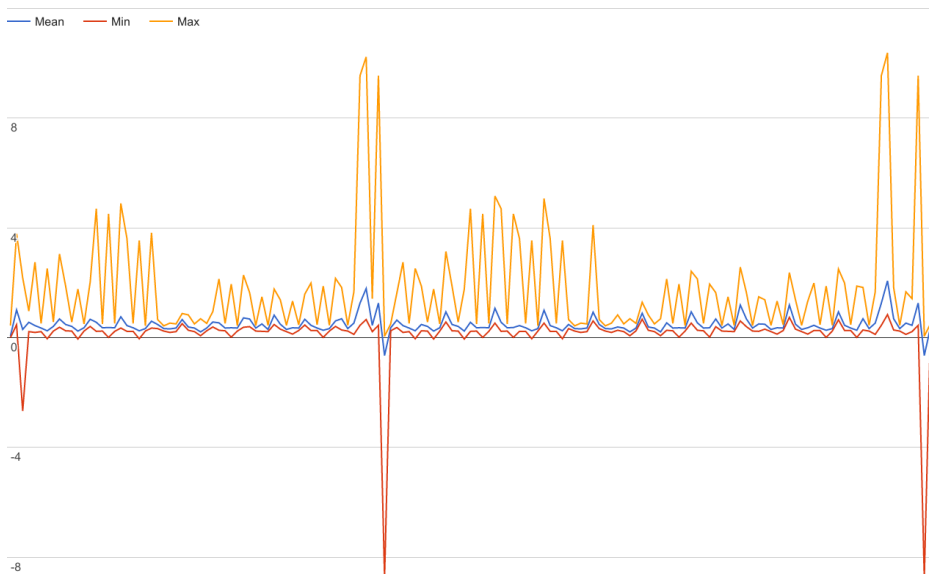


Figure 7.2: Average feature vector for bored, with the minimum and maximum values

indicates that different parts of the vector discriminates between emotional states. Kołakowska [2015] found that there were not one subset of features that could be used to discriminate between classes for all users, but that they were individual for each class. This statement could support how the graph looks, as there are not one subset of features that is different between all classes. However, keep in mind that these are only average vectors.

In figures 7.2 (bored), 7.3 (concentrated), 7.4 (confused), 7.5 (delighted), 7.6 (frustrated) and 7.7 (surprised), the mean vectors can be seen for each class. Additionally, the minimum and maximum value for each feature is included. None of the feature vectors include the pulse features. The mean, minimum and maximum values and their corresponding features can be found in appendix E.

Also, it might happen that even though participants have challenges in the current exercise, they become familiar with writing `console.log(`, and their reported emotional state is not reflected in how they type this specific keyword. On the other hand however, it might happen that their emotional state is indeed reflected in how they type the keyword. This is uncertain for the researcher.

As seen in table 5.3, two participants did not have any previous experience with programming, and their keystrokes may differ from those that have had at least half a year of practice. However, the researcher believes that in a real

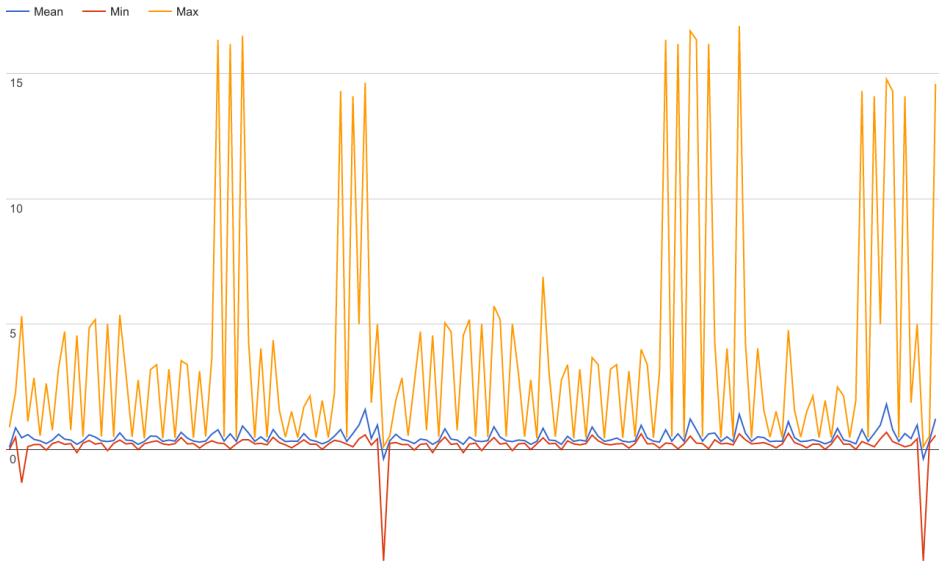


Figure 7.3: Average feature vector for concentrated, with the minimum and maximum values

life scenarios such a system would attract both experienced and inexperienced users. Although it should have been tested how these two participants affected the classification.

It should also be noted that the classifiers created might be too adapted to this dataset. However, this is not possible to test without a new data collection process.

7.1.3 Research question 1

In the universal multiclass classifier, the most promising results found were for the class delighted, with an average precision of 51.5% and an average recall of 52.5% across the KNN and SVM classifiers. When reducing the number of classes to four, delighted is still the class with highest accuracy with an average precision of 65% and an average recall of 53.5% across the KNN and SVM classifier. However, the kappa score is fairly low, below 0.30 for the full set of instances, and at 0.23 and 0.33 for the KNN and SVM classifier respectively in the subset classifier. The subset consisted of the classes bored, concentrated, delighted and frustrated.

In overall for these two classification tasks, the SVM classifier resulted in higher precision and recall than the KNN classifiers. For the multiclass classifier

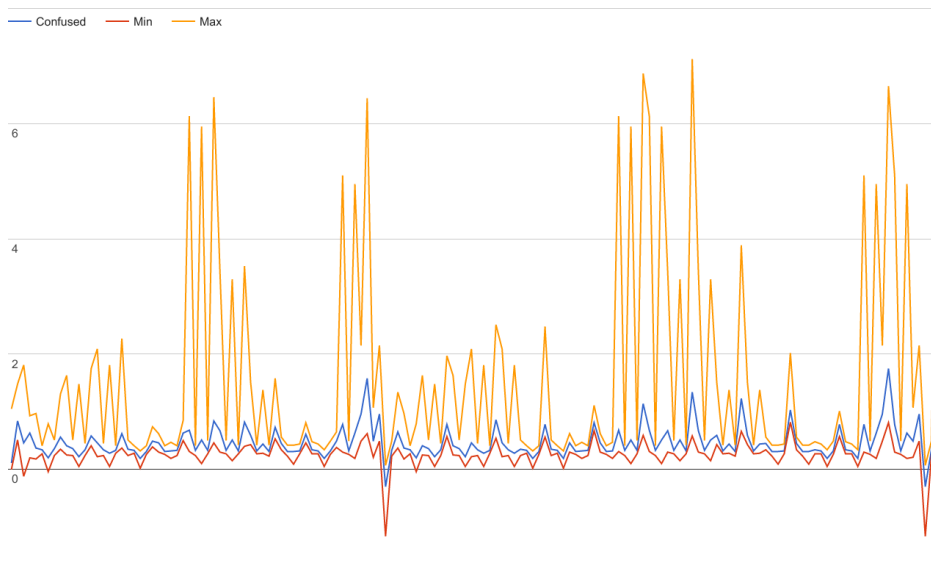


Figure 7.4: Average feature vector for confused, with the minimum and maximum values

tested on the complete set of instances, the average precision for the SVM was 39%, and the average recall was 42%. KNN, however, had an average precision of 34% and average recall of 37%. SVM also generated better results for the subset of classes with an average precision of 53% and an average recall of 52%. KNN had an average precision of 48% and an average recall of 42%. It should be noticed here that by reducing the precision from 17%, which would be random classification for 6 classes, to 50%, the chance to classify correctly have been doubled, from $1/6$ to $1/3$.

Limiting the number of classes had a positive impact on the precision and recall, however it also removes two classes hypothesised to be important in learning. The researcher speculates that few participants felt confused because the exercises presented did not introduce unfamiliar tasks. The low number of surprised instances might be because the feeling of surprise first came after the code had been run, i.e. when the participant sees if the result was successful or not. Such a case would suggest that participants felt surprised at the beginning the next exercise, but these explanations have not been subject to further investigation.

In the universal binary classifier, which also yielded the most promising results in Epp [2010], the precision was at 60% or higher for all classifiers except the SVM-concentrated and KNN-delighted classifiers. The recall ranged from 45% to 100%

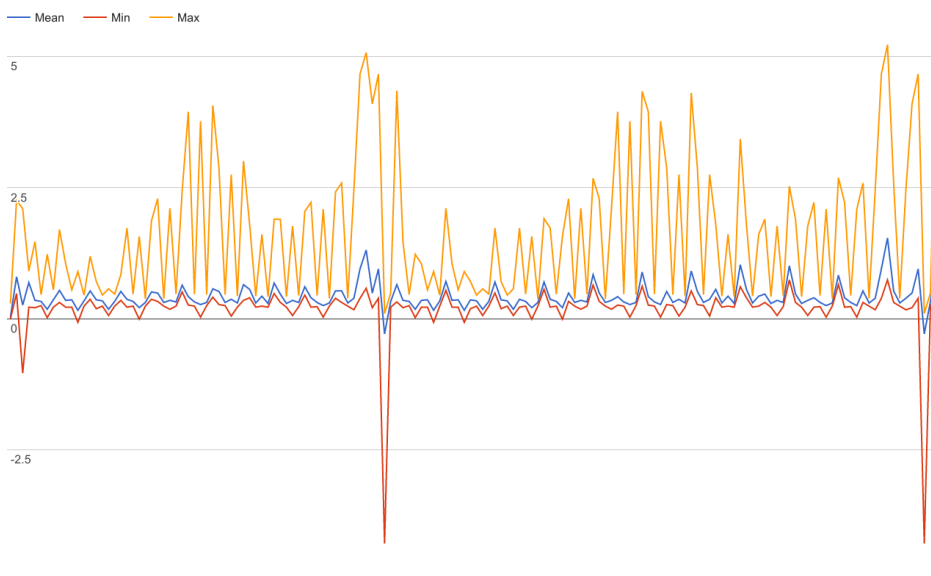


Figure 7.5: Average feature vector for delighted, with the minimum and maximum values

for all except the two classifiers already mentioned. Although these scores seem higher than that of the multiclass classifier, it is important to remember that the binary classifier only classify two classes, whereas the multiclass classifiers classify 4-6 classes. If left to chance alone a binary classifier should have a 50% chance to classify an instance correctly. The classifiers confidence can be seen on the kappa score in tables 6.2 (universal multiclass classifier), 6.4 (universal subset multiclass classifier) and 6.6 (universal binary classifier).

The binary classifier shows most promise for the classifiers KNN-bored, SVM-confused, SVM-frustrated and SVM-surprised, which had a class limit of 60, 22, 84 and 7 instances respectively. These had a kappa score of 0.22, 0.57, 0.65 and 1.00 respectively. It should be noted that all these classifiers used different hyperparameters. As such, the researcher has not found one specific classifier that is best suited for this task.

In the individual multiclass classifier each satisfactory classification is further apart than in the universal. This might be due to the limited dataset, as some classes counted 15 instances at the most. Of these 15 instances, only 1/3 were used in the validation set. As such, none of the individual multiclass classifiers yielded any results possible to infer a conclusion from.

For the individual binary classifiers, the number of instances for some classes

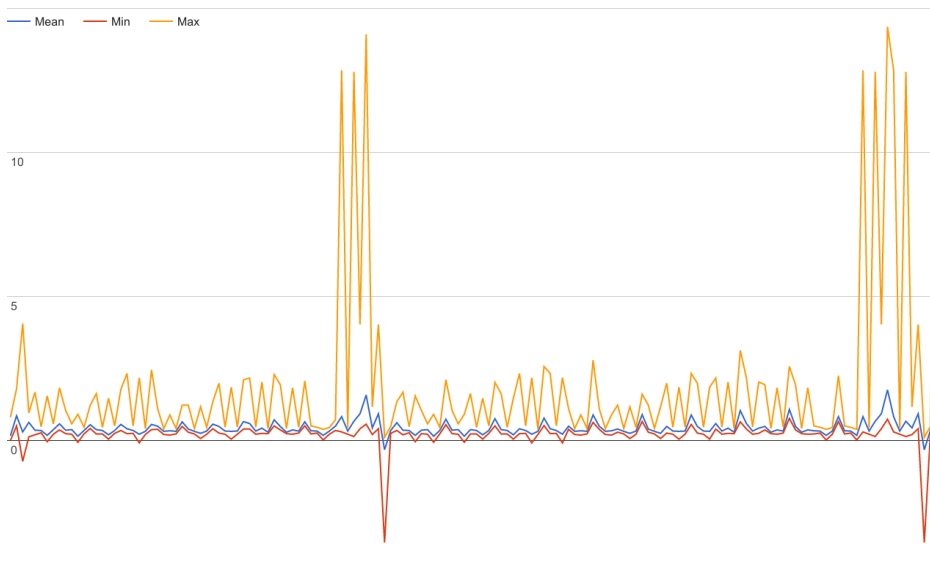


Figure 7.6: Average feature vector for frustrated, with the minimum and maximum values

and some participants were higher. The best being concentrated for participant A, which yielded a precision of 88% and a recall of 67% with the KNN classifier. The kappa score was at 0.67. Participants C had a promising result from the binary classifier SVM-frustrated, which had a precision of 84%, recall at 100%, and a kappa score of 0.53.

Is it possible to detect a learner's emotional state using keystroke dynamics from programming keywords? Not definitely, but the results are promising. The universal classifiers showed most promise, which may indicate that there are similarities in how typing rhythm is expressed for certain emotional states among several people. This information could be used to construct one or several average vectors that later could be adapted to individual people. However, this field needs more research, especially on feature selection and looking for features in more of the submitted code. Kołakowska [2015] only used short polish word, long enough for either a digraph or trigraph alone. Similarly it could be possible to use shorter patterns to find shorter sequences that are used in several keywords, to get data from words that are both familiar and unfamiliar to the user.

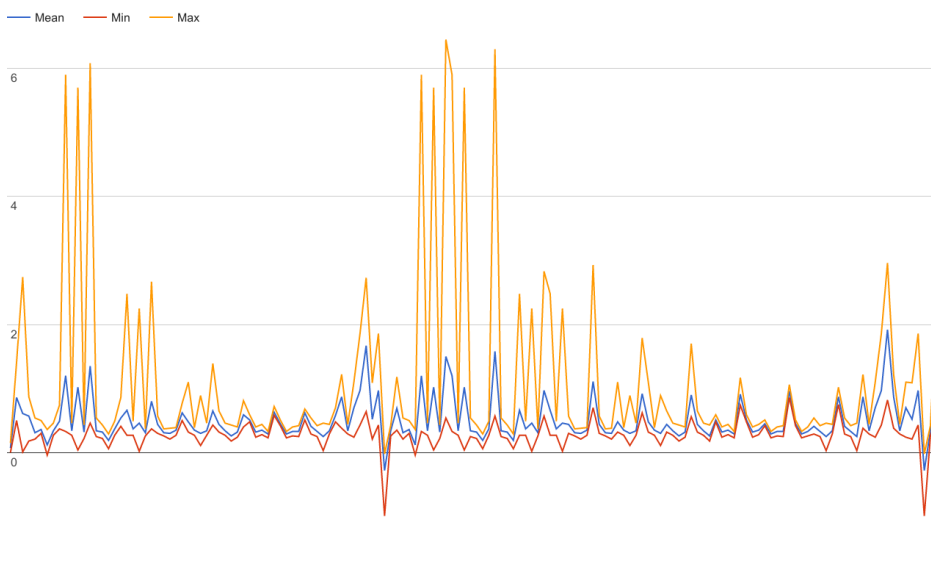


Figure 7.7: Average feature vector for surprised, with the minimum and maximum values

7.1.4 Research question 2

The dataset used for this research question is even more limited than the superset used for the first research question. This needs to be taken into account when answering the posed question. To answer this question, the same feature vectors was classified with and without pulse the features, using the same hyperparameters.

For the multiclass classifiers the feature vectors without pulse features performed equal or better than those with pulse features, however only marginally better. The most promising result was for the class concentrated, both for the full set and subset classifier. The individual multiclass classifier for participant H yielded a kappa score of 0.08 for KNN and -0.12 for SVM. This classifier was the only out of the five participants that got a positive kappa score, and thus not much confidence can be given to this result.

The individual binary classifier SVM-frustrated for participant D yielded a precision of 90% for the class frustrated, 100% recall for frustrated instances and 75% recall for the class “other”. The kappa score was 0.55, which is fairly good. The features without pulse yielded a precision of 68%, recall at 50% for both classes and a kappa score of 0, however, which is equal to random and thus not good. The second best kappa score was 0.36 for participant H’s KNN-frustrated

classifier. For participant H, there were no differences between the classifiers that used the pulse features and the ones who didn't.

The approach taken in this research, increasing the number of features from 153 to 156, yields that the three extra pulse features were not given much weight. Thus it was not surprising that there was not much difference in accuracy. In fact, the pulse features might make the vectors less accurate as they have not been adapted to participants' resting pulse.

Will detecting a learner's pulse yield a better classification than typing rhythm alone? With the information presented in this thesis, it is not possible to determine if the pulse increase or decrease the classifiers accuracies. To answer this question, steps should be taken to see how large of a factor the pulse is in determining the emotional state, versus that of the keystrokes alone. Additionally, different methods to merge the pulse data with the keystroke dynamics should be tested, either giving those features more weight, experimenting with other pulse features, or classify pulse features alone and merge the results, as proposed by KM et al. [2015].

7.1.5 Is it possible to determine a learner's emotional state non-intrusively in a programming tutorial?

This study alone can't give an definitive answer to this question, but the results from this study, and the studies done by Epp [2010] and Kołakowska [2015] show promise for such a possibility. There seems to be information in how people type that can be used to detect emotions. However, keystroke dynamics alone is probably not the best solution. As mentioned by Villani et al. [2006], typing rhythm can be affected by the environment, especially the keyboard. Thus, keystroke dynamics are prone to large uncertainties. Teh et al. [2013] stated that keystroke dynamics can not solely be relied on as a measure of authentication, but combined with other methods it will create a stronger authentication system. Hence, the researcher sees the necessity of combining this information with other sources, such as audio and video, to correctly assess a person's emotional state.

7.2 Contributions

In this section, the contributions from the study are summarised.

This research has tried to classify three emotional states that previously have not been classified using keystroke dynamics. These were concentrated, confused and delighted. The other three emotional states, bored, frustrated and surprised, have been tested, but not in the context of a programming tutorial. Most promising was the results from the universal binary classifier, with precisions at 24%, 79% and 100% using the SVM classifier for concentrated, confused and delighted

respectively, with kappa scores at 0, 0.57 and 1. The KNN classifier yielded a precision of 60%, 71% and 80%, with kappa scores at 0.20, 0.43 and 0.29. The highest score was for the class frustrated, with precision of 74% and 85% for KNN and SVM respectively, and kappa scores of 0.45 and 0.65 respectively.

Although Kołakowska [2016] sought to assess if a student was stressed or not during a programming exercise, no attempt has previously been made to assess a student's emotional state looking at how they write code. This study has shown that there is potential for such emotion recognition in programming.

This is the first time SVM has been used as a classification algorithm to detect emotional states from keystroke dynamics. Using this algorithm yielded slightly better results than those of KNN for most classes and classifiers. However, further work is needed on feature selection.

As a part of this thesis, a tutorial web application was created, which can be used to collect and store keystroke information. This approach differed from those of Epp [2010] and Kołakowska [2015], which had a background process running. The application proved successful in collecting data, and can further be developed to adapt to user's emotional state.

7.3 Future work

Moving forward on the findings of this thesis, and its ultimate goal of creating an adaptive programming tutorial, the researcher proposes these directions forward:

- Collect a large dataset from first year students learning JavaScript, or any other programming language. A large course, e.g. IT2805 Web technologies with its 200 students, is a suiting environment to collect large amount of data over a longer period of time.
- Use the emotional states presented in this thesis to discover which of the four quadrants presented in the model by Kort et al. [2001] (as seen in figure 2.3), the students are in. This can be used to give appropriate feedback during the learning cycle.
- Integrate an automatic pulse detector, which can detect pulse e.g. for each keystroke or key event, giving the pulse feature higher fidelity, and also allowing the correct pulse to be assigned to a substring of the code.
- In an adaptive environment, incorporate feedback from the compiler, together with information on the user's emotional state, to better determine how well the learner knows a certain subject.
- Use the integrated web camera to read a user's pulse, this must then be able to follow the user's head while in motion.

- Explore other non-intrusive data sources, such as mouse movements, facial recognition and eye tracking using the webcam.
- Find out if the time spent by the user before typing can be used to help detect their emotional state.
- Investigate other factors than emotional state, e.g. attention span, to see if this give a better indication on the learner's state.
- Account for the fact that users change their typing rhythm over time, as partially stated by Longi et al. [2015].
- Further develop the exercises presented in this thesis to induce emotional states, such as boredom or delight.

Appendices

Appendix A

Exercises

In the following sections are every exercise presented, as they were used in Adapt (see section 4.1). Each exercise consists of a title (presented in the section title after the exercise id), curriculum, exercise text, hint, error message, expected results, and a flag indicating if the exercise was timed or not.

A.1 Exercise 0: Thank you for participating

Curriculum "Welcome! Please read the exercise text before you start typing:)"

The first thing you need to know is how to output data. In JavaScript this is done using the function `console.log()`. Whatever is inside the two parentheses will be printed to the console. Usually in this tutorial you will be asked to output a string to the console (to the right of the code editor). A string is encapsulated by two ""quotes"".

If you have any feedback, please write to me at thorhb@stud.ntnu.no, or use the feedback form below the hint section.

May the odds be ever in your favour!"

Exercise text To start the tutorial, use the `console.log` function to print the string "I accept" to the console.

Doing so, you agree that I can store the keystroke dynamics provided by you while typing in the web editor and the information you provide in the forms presented during the tutorial.

Hint Whatever you write between the two parentheses will be printed to the console.

If you have several strings you can concatenate them by using the + operator, or a comma. Using a comma will add a space between the two strings.

Error message In order to proceed, you have to print "I accept" to the console.

Expected result I accept

Timed False

A.2 Exercise 1: JavaScript's building blocks

Curriculum In JavaScript, we operate with three basic data types that make up the fundamental building blocks in the language.

These three are:

- Strings: Any grouping of words or numbers surrounded by single `'...'` or double `"..."` quotes.
- Numbers: Any number, including decimals, without quotes, e.g. 13, 3.14.
- Boolean: Either true or false, without quotations.

We have arrays and objects also, but we will focus on the fundamentals first, and visit them later.

Exercise text Print string "JavaScript", the number 42 and the boolean value true (without quotes) to the console using one `console.log` statement for each data type.

Hint Remember:

- Strings are written with single or double quotes
- Numbers are written without quotes
- Boolean values can be either true or false (no caps), and are written without quotes.

Error message Did you write the values inside a `console.log` function?

Expected result Javascript 42 true

Timed False

A.3 Exercise 2: Quoteception

Curriculum Sometimes we want to use quotes inside a string. What do we do then? A string must start and end with the same type of quote, and it doesn't matter which one. So if you need to write Thomas' bike, you can do that by encapsulating the string in double quotes, leaving the single quote open to other useful things. I.e.: "Thomas' bike".

Exercise text Print the following string to the console : Nicholai's name isn't Thea

Hint A string must start and end with the same type of quotes.

Error message That wasn't the expected output

Expected result Nicholai's name isn't Thea

Timed False

A.4 Exercise 3: Concatenate strings

Curriculum It is useful to combine two strings, or a string and a number, e.g. when you want to greet a user by their name.

To do this, you can use the + operator between two strings. Writing `console.log("Hi" + "Thor")` would print "HiThor" to the console.

As you see, there is no space between the two strings. You can solve this by adding a space in one of the two strings, or use a comma between the two instead, like this: `console.log("Hi", "Thor")`. This will log "Hi Thor" to the console.

Exercise text Combine and print the two strings "Hi there, " and "Buffalo Bill" to the console.

Hint You can either use the + operator or a comma (,) between two strings to concatenate (combine) them.

Error message Did you concatenate the two strings, and did you remember the comma?

Expected result Hi there, Buffalo Bill

Timed False

A.5 Exercise 4: Numbers

Curriculum A quick recap on numbers: They are all integers and decimals, and are written without quotes.

Decimal numbers are written with a period/punctuation mark (`.`), and not a comma (`,`).

You can use arithmetic operations on them. This will be introduced in the following exercises.

So far you only need to remember the difference between strings and numbers.

Exercise text Print the decimal 42.4 to the console.

Hint Numbers are written without quotes.

If you print `"42"` to the console, you will log a string, and not a number.

Error message Decimal numbers are written with a period (`.`), not comma (`,`).

Expected result 42.4

Timed False

A.6 Exercise 5: Arithmetic operations

Curriculum Code is a language written on logic and math. But don't worry, math does not need to be your strong suit, basic arithmetic takes you a long way. The most used operators are:

- `+` adds two numbers
- `-` subtracts one number from another
- `/` divides one number with another
- `*` multiplies two numbers

Exercise text Inside four console.log statements, one for each operation, do the following:

- * Add 10 and 3.5
- * Subtract 1977 from 2017
- * Divide 200 by 4
- * Multiply 2 with 32

Hint Numbers are written without quotes. Decimals are written with a period (.) symbol, not comma (,).

Error message Did you know that the first Star Wars movie was released 40 years ago?!

Expected result 13.5 40 50 64

Timed False

A.7 Exercise 6: (Arithmetic + 5) / 2

Curriculum Arithmetic in JavaScript is controlled the same way as regular arithmetic. Multiplication and division have a higher precedence than addition and subtraction.

As with regular math, you can use parentheses to control when a calculation is made, e.g. if you want the addition to happen before multiplication.

Exercise text Use parentheses on the calculation $4 + 2 * 100$, so that the answer becomes 600 and not 204. Print the answer to the console.

Hint The order of calculation is as follows:

- Calculate parentheses
- – Multiplication and division, calculated left to right
- – Addition and subtraction, calculated left to right.

Error message Did you encapsulate $4 + 2$ in parentheses?

Expected result 600

Timed False

A.8 Exercise 7: Concatenate strings and numbers

Curriculum Earlier we concatenated two strings, but you can also concatenate a string and a number.

Perhaps you noticed that the `+` operator was used both in concatenation and in adding numbers.

JavaScript is a weakly typed language, meaning that the compiler will guess the data type. If one of the data types of either side of the `+` operator is a string, the result will be a new string. E.g. `"Age: " + 25` will yield the string `"Age: 25"`.

Exercise text Concatenate the number 42 with the string `"is the answer to life the universe and everything"` and print the result to the console.

Hint If you try to add `"4"` and 2, you will be given the string `"42"`.

Error message Did you write the string correctly?

Expected result 42 is the answer to life the universe and everything

Timed False

A.9 Exercise 8: Boolean type

Curriculum So you remember that a boolean type can be either true or false. At least I assume you do:) Boolean types can be used to if you need to say "yes" or "no" about something, e.g. if a constraint is fulfilled E.g. If you were to write `3 < 4` you would get false in return, as 3 is not greater than 4. When comparing two numbers, you can use `<` (less than), `>` (greater than), or `==` (equal). Two equal signs are used, as one `=` is used to assign a value to a variable.

Exercise text Check if 42 is greater than 13. Print the result to the console.

Hint The `>` operator means "greater than" as it says that the value on the left is greater than the one on the right. `<` means "less than", and `==` means "equal". The result from such an comparison will either be true or false.

The result should be either true or false.

Error message Did you use `>` and not `<`?

Expected result True

Timed False

A.10 Exercise 9: Comparing data types

Curriculum In the previous exercise we compared two numbers with the help of the "greater than" operator `>`. If you were confused, here is a short recap:

- `X > Y` means: Is X is greater than Y?
- `X < Y` means: Is X less than Y?
- `X == Y` means: Is X equal to Y?

This can be used on numbers, and on strings. How, you may ask. Each character in a string is saved as a number, thus we can add all the character's numbers and see which is larger. Note that upper case letters have a lower value than their small caps letter counterparts.

Exercise text Check if the string "AAA" is equal to "aaa". Print the result to the console

Hint The `>` operator means "greater than" as it says that the value on the left is greater than the one on the right. `<` means "less than", and `==` means "equal". The result from such an comparison will either be true or false.

Error message Did you use two equal signs (`==`)?

Expected result False

Timed False

A.11 Exercise 10: Variables

Curriculum So far we have only printed our data directly to the console. But what if we want to store the data for a while, so that we can reference it several times?

Enter variables! A variable can hold data, like strings, numbers and booleans. JavaScript have three different variable keywords:

- `var`: the old variable keyword, for any use case (ish)
- `let`: a variable that can be changed later
- `const`: a constant that can't be changed later

ES6 (the new JavaScript standard) uses only `let` and `const`. The rule for what to use when is: Does the value of the variable need to change? Use `let`. Otherwise you should use `const`.

A variable is declared as this: `let myVariable = "value"`.

`myVariable` can be any name you want to give your variable (it should describe the value). The name must start with an underscore or a letter. The value can be either a string, number, boolean, array, object or function (more on that later).

Exercise text Declare a constant and name it `year`. Give the constant the value of 1977. Print the constant to the console. You should reference the variable, and not simply print 1977.

Hint You reference the variable/constant by writing it's name, without any quotes. Adding quotes around the name would make it a string, and not a reference. The variable name is only a reference to the value. Thus the same value can be used several places, without specifically writing the value.

Error message As I'm sure you remember, 1977 was the year the first Star Wars movie was released, and hence a very important year to all.

Expected result 1977

Timed False

A.12 Exercise 11: Const vs let

Curriculum The difference between `const` and `let` can be a bit confusing. Remember that constants can't be changed later. You may think: Why would I even use a constant? It might be obvious later on, but they are safer from a data integrity perspective, as they can't change, and will stay the same. Sometimes you need to change the variable though, and then we have `let`.

If you try to give a constant another value later in your code, you will get an error. That will not happen with `let`.

Exercise text Declare a `let` variable, and name it `height`. Give it the value of 153. On the next line print the `height` variable. Then, on the third line write `height`, and give it the value of 157. It is important that you don't write `let` in front of the second `height`, as you will try to declare a new variable, and not change the value of the previous. Then print `height` to the console again. See how the value changes?

Hint You assign a new value to an already existing variable by writing the variable name, without `let` or `var` in front.

Error message Did you give the `height` variable a new value?

Expected result 153 157

Timed False

A.13 Exercise 12: Arrays

Curriculum So far we have talked about three data types. Now we will introduce another, called array. An array is a list of elements, that can be either strings, numbers or boolean, other arrays or objects.

An array is declared by two square brackets `[]`. E.g. if you have an array named `birds`, it would be written: `const birds = ["Ducks", "Eagles", "Hummingbird"]` Each element is separated by a comma `,`. Arrays can be used to store information in a sequential way (this will be clear later on).

Exercise text Declare a constant and name it `tasks`. Then assign it the value of an array, with the elements `"Complete this tutorial"`, `"Cook dinner"` and `"Take out the trash"`. Print the array to the console.

Hint An array is a list of data types, which are separated by commas. The list may contain strings, numbers, boolean values, variables, other lists and objects. You declare a list by encapsulating the content in square brackets (`[]`). In the curriculum objects were mentioned. This tutorial will as of now not cover objects, but you'll find more information about them at [codecademy.com](https://www.codecademy.com).

Error message Did you add all the elements to tasks?

Expected result `["Complete this tutorial", "Cook dinner", "Take out the trash"]`

Timed False

A.14 Exercise 13: Get the *n*th element of an array

Curriculum What do we do if we want the 3rd element of an array? Well, JavaScript has a solution for that!

But first, we need to learn that arrays in JavaScript (and other programming languages) starts at 0, not 1. So the first element in an array is the 0th element. Say we want the 3rd element in the array `birds = ["Ducks", "Eagles", "Hummingbird"]`, meaning you want "Hummingbird". By typing `birds[2]`, we will get "Hummingbird" in return.

Exercise text Declare a constant and name it `languages`. Then assign it the value of an array, with the elements "Python", "Java" and "JavaScript". Print the first and second element to the console. Use one `console.log` function per element

Hint You print a specific element of an array by writing `array[i]`, where `array` is the name of that array, and `i` is the index. Remember that the array's first element is at index 0.

Error message Did you use the correct indexes, and did you use two `console.log` statements?

Expected result `Python Java`

Timed False

A.15 Exercise 14: Get the last element of an array

Curriculum But then, what do we do when we want the last element of an array, and we don't know how long the array is? Arrays have a property that returns the numbers of elements in that array. Using `birds.length` on `birds = ["Ducks", "Eagles", "Hummingbird"]` will return 3, because there are three elements.

Of course, the last element have the index 2, so to get the last element we have to write `birds[birds.length-1]`, which will return "Hummingbird".

Exercise text Declare a constant and name it `superheroes`. Then assign it the value of an array, with the elements "Superman", "Wonder Woman" and "Batman". Print the last element to the console using the array's length function.

Hint You can get the length of an array, i.e. the number of elements in the list, by writing `array.length`, where `array` is the name of that array.

Remember that the array's first element is at index 0. If you try to get the element at `array[array.length]` you will get the next element after the last one, and that one doesn't exist.

Error message Did you use the correct index?

Expected result Batman

Timed False

A.16 Exercise 15: Loops

Curriculum Say that you want to print all the numbers from 1 to 10 to the console. That can be done by writing 10 `console.log` functions. If you want to print all numbers from 1 to 1000, that is a lot of work! Enter for loops. A for loop is a loop that iterates until some requirement is fulfilled. For each iteration, the index updates to progress the loop. A for loop is written: `for (let i = 1; i < 11; i++) ...`. The for loop takes three parameters, the first being the initial index (let `i = 0`), the next being when the loop should end (`i < 11`) which in this case the loop will continue until `i` is less than 11, which is 10. The last parameter is how the index should be updated. Writing `++` after a variable will update the variable with 1.

If we would print the index to the console with this for loop, we would get every number from 1 to and including 10. Note that in the explanation above I used ... to denote where the code block goes.

There is one important thing you need to know: Your loop needs to be terminated, otherwise it will go on forever, eventually using up all your RAM. Make sure that you set an condition that can be fulfilled. E.g. making the loop run as long as i is greater than 1, and i is ever increasing, you will find yourself in trouble.

Exercise text Using a for loop, print every other number from 2 up and to 10 to the console.

Hint In stead of using $i++$ to update the index by one, you need to update it by 2 for each iteration. In order to add a number to a variable, in stead of just replacing the value, you need to write $+=$, meaning that it should take the variable's value, and add whatever number comes after.

The code that you want to run should be located inside the curly brackets following the for loop.

Error message Did you update the index correctly?

Expected result 2 4 6 8 10

Timed False

A.17 Exercise 16: Factorials

Curriculum The factorial of a non-negative integer n is the product of all positive integers less than or equal to n .

E.g.: $5! = 5 * 4 * 3 * 2 * 1 = 120$.

This exercise is timed.

Exercise text Using a for loop, sum up the factorial of the number 10. Print the result to the console.

Hint Remember to end the loop in the correct place.

Error message Did you add the products together?

Expected result 3628800

Timed True

A.18 Exercise 17: Multiples of n

Curriculum Do you remember the multiplication table? With programming you don't have to!

This exercise is timed.

Exercise text Print every number from 0 (not 0) up to, and including, 60 that is a multiple of 3.

Hint A for loop might be the correct tool for this job.

Error message Did you start at 3?

Expected result 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60

Timed True

A.19 Exercise 18: Looping through an array

Curriculum We now know how to print the index to the console. We also know how to get the *n*th element of a list. If you don't remember, here's a refresher:

`array[n]` will get you the *n*th element of an array. We can use this knowledge to print every element of the array to the console by writing: `for (let i = 0; i < array.length; i++) console.log(array[i])`

Exercise text Declare a constant and name it `colors`. Then assign it the value of an array, with the elements "Red", "White" and "Blue". Using a for loop, print each element to the console, but add an asterisk (*) in front of each element. The result should look like a list.

Hint Each element in this array is a list, you can thus concatenate the string "*" and the array element. You get the value of each array element by writing `array[i]`, where `array` is the name of the array, and `i` is the index of that list element.

Remember that arrays start with index 0, and that the last element is the array's length minus 1.

Error message Did you concatenate the two strings correctly?

Expected result * Red * White * Blue

Timed False

A.20 Exercise 19: What if?

Curriculum Sometime we may want to do action based on a condition, e.g. if the user's age is above or below 18 decide if they can buy beer or not. For these cases we have the `if` statement, which checks if a condition is true or false. If statements are written as this: `if (condition) ...`

The condition can e.g. be `age >= 18`. If the condition evaluates to true (yes, my age is greater than or equal to 18), then the code within the curly brackets `()` will be executed, otherwise it will be skipped.

Exercise text Declare a let variable, name it `weather` and give it the value of `"sun"`. Then create an `if` statement which checks if the value of `weather` is equal to the string `"sun"`. Inside the code block, print the string `"It's sunny outside"` to the console.

Hint Within the two parentheses you'll find the evaluation. When you are evaluating two data types, you use two equal signs. If the two values are identical, the expression evaluates to true.

Error message Did you use two equal signs (`==`) in the conditional?

Expected result It's sunny outside

Timed False

A.21 Exercise 20: What else?

Curriculum You want to check if some condition is true, but it's not, so you want to run your backup plan? Then you can add an else statement after the if. If the if statement evaluates to false, the code within the else block will be run. This is written: `if (condition) code else code`

Exercise text Declare a let variable and name it `weather`, and give it the value of `"rain"`. Then create an if statement which checks if the value of `weather` is equal to the string `"sun"`. Inside that code block, print the string `"It's sunny outside"` to the console.

Then, after the if code block, add an else block, where you print `"I'm not sure what weather it is outside"` to the console. If the code is correct, `"I'm not sure what weather it is outside"` will be printed to the console.

Hint The else block catches all cases that are not covered by the preceding if blocks. else is written after the last curly bracket of the if block.

Error message Have you written the correct expression inside the parentheses?

Expected result `I'm not sure what weather it is outside`

Timed `False`

A.22 Exercise 21: What else if?

Curriculum Of course it's often not enough with just checking for one specific case. In the previous exercise, we could have caught that it was rain, and printed that to the console. To add several cases, you can add what's called "else if".

Similar to if statements, they take an conditional that will evaluate to true or false. After the initial if statement, write `"else if (conditional) ... "`. This can then be followed by an else block, as in the previous exercise.

Exercise text Declare a let variable and name it `weather`, and give it the value of `"rain"`. Then create an if statement which checks if the value of `weather` is equal to the string `"sun"`. Inside that code block, print the string `"It's sunny outside"` to the console.

Then, after the if code block, add an else if block that checks if the value of `weather` is equal to `"rain"`. Inside that block, print `"It's raining outside"` to the console.

Hint The else if block comes after the last curly bracket if block, and needs a conditional (some expression that can evaluate to true or false) inside the following parentheses.

Error message Have you written the correct expression inside the parentheses?

Expected result It's raining outside

Timed False

A.23 Exercise 22: What if and if?

Curriculum In some cases, we only want to execute code if two or more conditionals are true. One way to solve this is to nest if statements inside each other. Another approach is to use the comparison operator `&&`, meaning "and".

When using `&&`, the expression will only evaluate to true if both (or all) parameters are true. E.g. `(X && Y) ...` will only be true if both X and Y are true. If one is false, the code will not be executed.

Exercise text Declare one let variable and name it `weather`, and give it the value of "rain". Then declare another let variable and give it the name `umbrella` and the value of true.

Then write an if statement that checks if `weather` is equal to "rain" and `umbrella` is true. Inside the if statement, print the string "You can safely walk outside" to the console.

Hint In the if statement, you don't have to write `"umbrella === true"`, a boolean can only be true or false, so you can simply ask `"umbrella"`, which will return the value of the variable.

Error message Have you written the correct expression inside the parentheses?

Expected result You can safely walk outside

Timed False

A.24 Exercise 23: What if or if?

Curriculum There are also cases where we want to execute code if one of two requirements are true. In that case, we have the operator `—`, meaning "or". If you are using the or operator, only one of the parameters have to be true for the code to be executed.

E.g. `(X — Y) ...` will be executed if X is true or Y is true, or both, but not if both are false.

Exercise text Declare one let variable and name it `weather`, and give it the value of "sun". Then declare another let variable and give it the name `umbrella`, and the value of false.

Then write an if statement that checks if `weather` is equal to "sun" or if `umbrella` is true. Inside the if block, print the string "You can safely walk outside" to the console.

Hint If you are checking if a boolean value is true, it is enough to write the variable's name.

Error message Have you written the correct expression inside the parentheses?

Expected result You can safely walk outside

Timed False

A.25 Exercise 24: How about no?

Curriculum Sometimes we want to execute code if a constraint is not fulfilled. By placing a exclamation mark (!) before a parameter, you will negate the boolean value.

E.g. if you evaluate the boolean value of the variable `let name = "Thor"`, you will get true in return. However, if you evaluate `!name`, you will get false in return.

Exercise text Declare a constant, name it `user` and give it the value of false. This variable plays the role of our web page's imaginary user, and that user is not registered.

Now write an if statement that checks if the user constant is false. If the user is false (i.e. the user is not registered), the string "You need to register" should be printed to the console.

Hint In an if statement, when we check if some variable or constant is true, we can simply write that variable or constant's name. When we want to check if the variable or constant is false, we need to negate it.

Error message Have you written the correct expression inside the parentheses?

Expected result You need to register

Timed False

A.26 Exercise 25: Functions

Curriculum So far we have written our code in one big pile of fun. That works well enough for small code blocks, but when the complexity of our application increases, we need to structure the code into functions. A function is a block of code that can be invoked elsewhere in your code. Using function you only have to write code once, even though it will be used several places. Which leads to less errors, and code that are easier to maintain.

You declare a function by writing the keyword function, followed by the function name and two parenthesis. Then you have two curly brackets, with the code block inside.

E.g. `function printAJoke()console.log('The cat's tie')` For a function to be run, it needs to be invoked. You do that by writing, outside of the function, the functions name, followed by two parenthesis. E.g. to invoke our function `printAJoke`, we would write `printAJoke()`.

Exercise text Declare a function with the name `helloFunction`. Inside the function, print the string "Hello function". Invoke the function.

Hint A function can also be declared as a variable, e.g. `const myFunction()code` This is purely preference.

Remember to invoke your function by writing the function name followed by two parentheses, e.g. `myFunction()`.

Error message Did you invoke the function?

Expected result Hello function

Timed False

A.27 Exercise 26: Functions that manipulate the data

Curriculum Wouldn't it be useful to do something with data inside the function? Do to so, we need to send the data to the function.

A function can take zero or more parameters, and do something with it. To do this, inside the parenthesis following the function name, write the name of your parameter. A parameter is like a variable, that is local to that function.

E.g.: `function printNumber(number)console.log(number). printNumber(22)` would print the number 22 to the console. The 22 that we send from the function call (where we invoke the function) is called arguments.

Exercise text Declare the function `multiplyWithTwo` that takes one parameter. The parameter name can be whatever, e.g. `number`. The function should multiply the parameter with 2 and print the result to the console. Invoke the function with the argument 16.

Hint Use the multiplication operator (`*`) to multiply a number with another.

Error message Did you multiply the parameter with two and print it to the console?

Expected result 32

Timed False

A.28 Exercise 27: Taking several parameters

Curriculum As mentioned, a function can take any number of parameters. Inside the parentheses following the function name, list all the parameters separated by commas (`,`).

It is a good idea to give the parameters meaningful names, so that you know what they mean. When you invoke the function, the arguments doesn't need to have the same name as the parameters, but the order is important! The first argument will be used as the first parameter, and so on.

Exercise text Declare the function `subtractTwoNumbers` that takes two parameters. The parameter names can be whatever, e.g. `number1` and `number2`. The function should subtract the second parameter from the first and print the result to the console. Invoke the function with the arguments 24 and 4.

Hint You invoke a function with several parameters as such: `myFunction(argument1, argument2)`.

In the function: `function myFunction(parameter1, parameter2) ... argument1 == parameter1, and argument2 == parameter2.`

If you change the order of the arguments, `argument2 == parameter1.`

Error message Did you subtract the right parameter, and print the result to the console?

Expected result 20

Timed False

A.29 Exercise 28: Returning the result from a function

Curriculum The function doesn't have to print the result to the console itself. It can return the result, so that other functions can use it. Instead of writing `console.log()` as we have so far, write `return`, followed by what you want to return.

E.g. `return parameter1 * 2.` You don't need parentheses around the return statement.

Exercise text Declare the function `divideTwoNumbers` that takes two parameters. The parameter names can be whatever, e.g. `number1` and `number2`. The function should divide the first number by the second and return the result.

Then invoke the function from within a `console.log` function with the arguments 512 and 64. This will print the result to the console.

Hint The return statement doesn't use parentheses, but you can only return one value. If you need to return several, store them in an array (not necessary for this exercise).

Error message Did you return the value to the `console.log` function?

Expected result 8

Timed False

A.30 Exercise 29: The beginning of the end

Curriculum In the following exercises you will be tested on the skills you have just acquired. The exercises are timed.

Exercise text Print the string "Let's begin" to the console to start the last exercises

Hint You shouldn't need a hint for this really.

Error message Did you print "Let's begin" to the console?

Expected result let's begin

Timed False

A.31 Exercise 30: Find the smallest number

Curriculum You don't need any new information to solve this exercise.

Exercise text Write a function that finds the smallest of two numbers. Invoke the function with the arguments 102 and 534 and print the result to the console.

Hint Use the if and else statements to check if one number is smaller than the other.

Error message Did you use the "less than" (<) operator?

Expected result 102

Timed True

A.32 Exercise 31: Cheesecake

Curriculum For this exercise we need to know about modulo, the % operator. Using % between two numbers will give you the remainder of dividing the two numbers. E.g. $5\%2$ will give us 1 because $5/2$ is equal to $2 + 2 + 1$. In this task you will be asked to see if a number is a multiple of another, meaning that the remainder should be 0.

E.g. if you want to find multiple of 6, you check `number%6 == 0`. If the remainder of number divided by 6 is 0, the number is a multiple of 6.

Exercise text Write a program that prints the numbers from 1 to 100. But for multiples of three print "cheese" instead of the number and for the multiples of five print "cake". For numbers which are multiples of both three and five print "cheesecake".

Hint Use `if`, `else if` and `else` statements to handle the different cases. And use `modulo` to see if a number is a multiple of another.

Error message Did you remember to print the numbers?

Expected result 1 2 cheese 4 cake cheese 7 8 cheese cake 11 cheese 13 14 cheesecake 16 17 cheese 19 cake cheese 22 23 cheese cake 26 cheese 28 29 cheesecake 31 32 cheese 34 cake cheese 37 38 cheese cake 41 cheese 43 44 cheesecake 46 47 cheese 49 cake cheese 52 53 cheese cake 56 cheese 58 59 cheesecake 61 62 cheese 64 cake cheese 67 68 cheese cake 71 cheese 73 74 cheesecake 76 77 cheese 79 cake cheese 82 83 cheese cake 86 cheese 88 89 cheesecake 91 92 cheese 94 cake cheese 97 98 cheese cake

Timed True

A.33 Exercise 32: Comparing arrays

Curriculum You don't need any new information to solve this exercise.

Exercise text Write a function that takes two array parameters and checks if they are identical. Return `true` if they are, and `false` otherwise. Invoke the function once with the arguments `["a", "b", "c"]` and `["a", "b", "d"]`, and once with the arguments `["a", "b", "c"]` and `["a", "b", "c"]`. Print the result to the console.

Hint You know this!

Error message You can do this! Have faith in yourself.

Expected result false true

Timed True

Appendix B

Application to Norwegian Centre for Research Data



MELDESKJEMA

Meldeskjema (versjon 1.4) for forsknings- og studentprosjekt som medfører meldeplikt eller konsesjonsplikt (jf. personopplysningsloven og helseregisterloven med forskrifter).

1. Intro		
Samles det inn direkte personidentifiserende opplysninger?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	En person vil være direkte identifiserbar via navn, personnummer, eller andre personentydige kjennetegn. Les mer om hva personopplysninger .
Hvis ja, hvilke?	<input type="checkbox"/> Navn <input type="checkbox"/> 11-sifret fødselsnummer <input type="checkbox"/> Adresse <input type="checkbox"/> E-post <input type="checkbox"/> Telefonnummer <input type="checkbox"/> Annet	NB! Selv om opplysningene skal anonymiseres i oppgave/rapport, må det krysses av dersom det skal innhentes/registreres personidentifiserende opplysninger i forbindelse med prosjektet.
Annet, spesifiser hvilke		
Samles det inn bakgrunnsopplysninger som kan identifisere enkeltpersoner (indirekte personidentifiserende opplysninger)?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	En person vil være indirekte identifiserbar dersom det er mulig å identifisere vedkommende gjennom bakgrunnsopplysninger som for eksempel bostedskommune eller arbeidsplass/skole kombinert med opplysninger som alder, kjønn, yrke, diagnose, etc.
Hvis ja, hvilke		NB! For at stemme skal regnes som personidentifiserende, må denne bli registrert i kombinasjon med andre opplysninger, slik at personer kan gjenkjennes.
Skal det registreres personopplysninger (direkte/indirekte/via IP-/e-post adresse, etc) ved hjelp av nettbaserte spørreskjema?	Ja <input checked="" type="radio"/> Nei <input type="radio"/>	Les mer om nettbaserte spørreskjema .
Blir det registrert personopplysninger på digitale bilde- eller videoopptak?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	Bilde/videoopptak av ansikter vil regnes som personidentifiserende.
Søkes det vurdering fra REK om hvorvidt prosjektet er omfattet av helseforskningsloven?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	NB! Dersom REK (Regional Komité for medisinsk og helsefaglig forskningsetikk) har vurdert prosjektet som helseforskning, er det ikke nødvendig å sende inn meldeskjema til personvernombudet (NB! Gjelder ikke prosjekter som skal benytte data fra pseudonyme helseregistre). Dersom tilbakemelding fra REK ikke foreligger, anbefaler vi at du avventer videre utfylling til svar fra REK foreligger.
2. Prosjektittel		
Prosjektittel	Detecting emotions during coding tutorials	Oppgi prosjektets tittel. NB! Dette kan ikke være «Masteroppgave» eller liknende, navnet må beskrive prosjektets innhold.
3. Behandlingsansvarlig institusjon		
Institusjon	NTNU	Velg den institusjonen du er tilknyttet. Alle nivå må oppgis. Ved studentprosjekt er det studentens tilknytning som er avgjørende. Dersom institusjonen ikke finnes på listen, har den ikke avtale med NSD som personvernombud. Vennligst ta kontakt med institusjonen.
Avdeling/Fakultet	Fakultet for informasjonsteknologi og elektroteknikk (IE)	
Institutt	Institutt for datateknologi og informatikk	
4. Daglig ansvarlig (forsker, veileder, stipendiat)		
Fornavn	Asbjørn	Før opp navnet på den som har det daglige ansvaret for prosjektet. Veileder er vanligvis daglig ansvarlig ved studentprosjekt.
Etternavn	Thomassen	
Silling	Assistant Professor	Daglig ansvarlig og student må i utgangspunktet være tilknyttet samme institusjon. Dersom studenten har ekstern veileder, kanbiveileder eller fagansvarlig ved studiestedet stå som daglig ansvarlig.
Telefon	73591839	Arbeidssted må være tilknyttet behandlingsansvarlig institusjon, f.eks. underavdeling, institutt etc.
Mobil	73591839	
E-post	asbjorn.thomassen@ntnu.no	NB! Det er viktig at du oppgir en e-postadresse som brukes aktivt. Vennligst gi oss beskjed dersom den endres.
Alternativ e-post	asbjorn.thomassen@ntnu.no	
Arbeidssted	NTNU/IE/IDI	

Adresse (arb.)	Sem Sælands vei 9	
Postnr./sted (arb.sted)	7034 Trondheim	
5. Student (master, bachelor)		
Studentprosjekt	Ja • Nei ○	Dersom det er flere studenter som samarbeider om et prosjekt, skal det velges en kontaktperson som føres opp her. Øvrige studenter kan føres opp under pkt 10.
Fornavn	Thor Håkon	
Etternavn	Bredesen	
Telefon	97629575	
Mobil		
E-post	thorhb@stud.ntnu.no	
Alternativ e-post	thor.hakon.bredesen@gmail.com	
Privatadresse	Maristuveien 2	
Postnr./sted (privatadr.)	7030 Trondheim	
Type oppgave	<ul style="list-style-type: none"> • Masteroppgave ○ Bacheloroppgave ○ Semesteroppgave ○ Annet 	
6. Formålet med prosjektet		
Formål	<p>Prosjektet skal se på om det er mulig å anslå en brukers følelser mens han/hun gjør programmeringsoppgaver vha. keystroke dynamics (tastatur) og puls (webkamera eller pulsmåler). Formålet er å gi en tilpasset progresjon (vanskelighetsgrad) i oppgavene til følelsene, for så å øke mestring og læring.</p> <p>Forsknings spørsmål 1: Kan emosjoner bedre detekteres vha nøkkelord enn alle ord i koden? Forsknings spørsmål 2: Kan deteksjon av puls forbedre klassifisering av følelser?</p>	Redegjør kort for prosjektets formål, problemstilling, forsknings spørsmål e.l.
7. Hvilke personer skal det innhentes personopplysninger om (utvalg)?		
Kryss av for utvalg	<input type="checkbox"/> Barnehagebarn <input checked="" type="checkbox"/> Skoleelever <input type="checkbox"/> Pasienter <input type="checkbox"/> Brukere/klienter/kunder <input type="checkbox"/> Ansatte <input type="checkbox"/> Barnevernsbarn <input type="checkbox"/> Lærere <input type="checkbox"/> Helsepersonell <input type="checkbox"/> Asylsøkere <input type="checkbox"/> Andre	
Beskriv utvalg/deltakere	Studenter/venner som er bekjente av meg	Med utvalg menes dem som deltar i undersøkelsen eller dem det innhentes opplysninger om.
Rekruttering/trekking	Eget nettverk/venner på universitetet	Beskriv hvordan utvalget trekkes eller rekrutteres og oppgi hvem som foretar den. Et utvalg kan trekkes fra registre som f.eks. Folkeregisteret, SSB-registre, pasientregistre, eller det kan rekrutteres gjennom f.eks. en bedrift, skole, idrettsmiljø eller eget nettverk.
Førstegangskontakt	Kontakt opprettes av undertegnede, i samtale med personen (eks.: jeg spør om en venn har lyst til å hjelpe meg med å bygge datasettet til prosjektet)	Beskriv hvordan kontakt med utvalget blir opprettet og av hvem. Les mer om dette på temasidene .
Alder på utvalget	<input type="checkbox"/> Barn (0-15 år) <input type="checkbox"/> Ungdom (16-17 år) <input checked="" type="checkbox"/> Voksne (over 18 år)	Les mer om forskning som involverer barn på våre nettsider.
Omtrentlig antall personer som inngår i utvalget	5	
Samles det inn sensitive personopplysninger?	Ja ○ Nei •	Les mer om sensitive opplysninger .

Hvis ja, hvilke?	<input type="checkbox"/> Rasemessig eller etnisk bakgrunn, eller politisk, filosofisk eller religiøs oppfatning <input type="checkbox"/> At en person har vært mistenkt, siktet, tiltalt eller dømt for en straffbar handling <input type="checkbox"/> Helseforhold <input type="checkbox"/> Seksuelle forhold <input type="checkbox"/> Medlemskap i fagforeninger	
Inkluderes det myndige personer med redusert eller manglende samtykkekompetanse?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	Les mer om pasienter, brukere og personer med redusert eller manglende samtykkekompetanse .
Samles det inn personopplysninger om personer som selv ikke deltar (tredjepersoner)?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	Med opplysninger om tredjeperson menes opplysninger som kan spores tilbake til personer som ikke inngår i utvalget. Eksempler på tredjeperson er kollega, elev, klient, familiemedlem.
8. Metode for innsamling av personopplysninger		
Kryss av for hvilke datainnsamlingsmetoder og datakilder som vil benyttes	<input type="checkbox"/> Papirbasert spørreskjema <input checked="" type="checkbox"/> Elektronisk spørreskjema <input type="checkbox"/> Personlig intervju <input type="checkbox"/> Gruppeintervju <input checked="" type="checkbox"/> Observasjon <input type="checkbox"/> Deltakende observasjon <input type="checkbox"/> Blogg/sosiale medier/Internett <input type="checkbox"/> Psykologiske/pedagogiske tester <input type="checkbox"/> Medisinske undersøkelser/tester <input type="checkbox"/> Journaldata (medisinske journaler)	<p>Personopplysninger kan innhentes direkte fra den registrerte f.eks. gjennom spørreskjema intervju, lester, og/eller ulike journaler (f.eks. elevmapper, NAV, PPT, sykehus) og/eller registre (f.eks. Statistisk sentralbyrå, sentrale helseregistre).</p> <p>NBI Dersom personopplysninger innhentes fra forskjellige personer (utvalg) og med forskjellige metoder, må dette spesifiseres i kommentar-boksen. Husk også å legge ved relevante vedlegg til alle utvalgs-gruppene og metodene som skal benyttes.</p> <p>Les mer om registerstudier her.</p> <p>Dersom du skal anvende registerdata, må variabeliste lastes opp under pkt. 15</p>
	<input type="checkbox"/> Registerdata	
	<input type="checkbox"/> Annen innsamlingsmetode	
Tilleggsopplysninger		
9. Informasjon og samtykke		
Oppgi hvordan utvalget/deltakerne informeres	<input checked="" type="checkbox"/> Skriftlig <input type="checkbox"/> Muntlig <input type="checkbox"/> Informeres ikke	<p>Dersom utvalget ikke skal informeres om behandlingen av personopplysninger må det begrunnes.</p> <p>Les mer her.</p> <p>Vennligst send inn mal for skriftlig eller muntlig informasjon til deltakerne sammen med meldeskjema.</p> <p>Last ned en veiledende mal her.</p> <p>NBI Vedlegg lastes opp til sist i meldeskjemaet, se punkt 15 Vedlegg.</p>
Samtykker utvalget til deltakelse?	<input checked="" type="radio"/> Ja <input type="radio"/> Nei <input type="radio"/> Flere utvalg, ikke samtykke fra alle	<p>For at et samtykke til deltakelse i forskning skal være gyldig, må det være frivillig, uttrykkelig og informert.</p> <p>Samtykke kan gis skriftlig, muntlig eller gjennom en aktiv handling. For eksempel vil et besvart spørreskjema være å regne som et aktivt samtykke.</p> <p>Dersom det ikke skal innhentes samtykke, må det begrunnes.</p>
10. Informasjonssikkerhet		
Hvordan registreres og oppbevares personopplysningene?	<input type="checkbox"/> På server i virksomhetens nettverk <input type="checkbox"/> Fysisk isolert PC tilhørende virksomheten (dvs. ingen tilknytning til andre datamaskiner eller nettverk, interne eller eksterne) <input checked="" type="checkbox"/> Datamaskin i nettverkssystem tilknyttet Internett tilhørende virksomheten <input checked="" type="checkbox"/> Privat datamaskin <input type="checkbox"/> Videoopptak/fotografi <input type="checkbox"/> Lydopptak <input type="checkbox"/> Notater/papir <input type="checkbox"/> Mobile lagringsenheter (bærbar datamaskin, minnepenn, minnekort, cd, ekstern harddisk, mobiltelefon) <input type="checkbox"/> Annen registreringsmetode	<p>Merk av for hvilke hjelpemidler som benyttes for registrering og analyse av opplysninger.</p> <p>Sett flere kryss dersom opplysningene registreres på flere måter.</p> <p>Med «virksomhet» menes her behandlingsansvarlig institusjon.</p> <p>NBI Som hovedregel bør data som inneholder personopplysninger lagres på behandlingsansvarlig sin forskningsserver.</p>
Annen registreringsmetode beskriv		Lagring på andre medier - som privat pc, mobiltelefon, minnepenne, server på annet arbeidssted - er mindre sikkert, og må derfor begrunnes. Slik lagring må avklares med behandlingsansvarlig institusjon, og personopplysningene bør krypteres.

Hvordan er datamaterialet beskyttet mot at uvedkommende får innsyn?	Det skal brukes private datamaskiner. Informasjonen innhentes via en nettside. Informasjonen lagres i en database kun undertegnede har tilgang til.	Er f.eks. datamaskintilgangen beskyttet med brukernavn og passord, står datamaskinen i et låsbart rom, og hvordan sikres bærbare enheter, utskrifter og opptak?
Samles opplysningene inn/behandles av en databehandler (ekstern aktør)?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	Dersom det benyttes eksterne til helt eller delvis å behandle personopplysninger, f.eks. Questback, transkriberingsassistent eller tolk, er dette å betrakte som en databehandler. Slike oppdrag må kontraktreguleres.
Hvis ja, hvilken		
Overføres personopplysninger ved hjelp av e-post/Internett?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	F.eks. ved overføring av data til samarbeidspartner, databehandler m.m.
Hvis ja, beskriv?		Dersom personopplysninger skal sendes via internett, bør de krypteres tilstrekkelig. Vi anbefaler for ikke lagring av personopplysninger på nettskytjenester. Dersom nettskytjeneste benyttes, skal det inngås skriftlig databehandleravtale med leverandøren av tjenesten.
Skal andre personer enn daglig ansvarlig/student ha tilgang til datamaterialet med personopplysninger?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	
Hvis ja, hvem (oppgi navn og arbeidssted)?		
Utleveres/deles personopplysninger med andre institusjoner eller land?	<input checked="" type="radio"/> Nei <input type="radio"/> Andre institusjoner <input type="radio"/> Institusjoner i andre land	F.eks. ved nasjonale samarbeidsprosjekter der personopplysninger utveksles eller ved internasjonale samarbeidsprosjekter der personopplysninger utveksles.
11. Vurdering/godkjenning fra andre instanser		
Søkes det om dispensasjon fra taushetsplikten for å få tilgang til data?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	For å få tilgang til taushetsbelagte opplysninger fra f.eks. NAV, PPT, sykehus, må det søkes om dispensasjon fra taushetsplikten. Dispensasjon søkes vanligvis fra aktuelt departement.
Hvis ja, hvilke		
Søkes det godkjenning fra andre instanser?	Ja <input type="radio"/> Nei <input checked="" type="radio"/>	F.eks. søke registerere om tilgang til data, en ledelse om tilgang til forskning i virksomhet, skole.
Hvis ja, hvilken		
12. Periode for behandling av personopplysninger		
Prosjektstart	15.03.2017	Prosjektstart Vennligst oppgi tidspunktet for når kontakt med utvalget skal gjøres/datainnsamlingen starter.
Planlagt dato for prosjektslutt	15.05.2017	Prosjektslutt: Vennligst oppgi tidspunktet for når datamaterialet enten skal anonymiseres/slettes, eller arkiveres i påvente av oppfølgingsstudier eller annet.
Skal personopplysninger publiseres (direkte eller indirekte)?	<input type="checkbox"/> Ja, direkte (navn e.l.) <input type="checkbox"/> Ja, indirekte (bakgrunnsopplysninger) <input checked="" type="checkbox"/> Nei, publiseres anonymt	NBI Dersom personopplysninger skal publiseres, må det vanligvis innhentes eksplisitt samtykke til dette fra den enkelte, og deltakere bør gis anledning til å lese gjennom og godkjenne sitatet.
Hva skal skje med datamaterialet ved prosjektslutt?	<input checked="" type="checkbox"/> Datamaterialet anonymiseres <input type="checkbox"/> Datamaterialet oppbevares med personidentifikasjon	NBI Her menes datamaterialet, ikke publikasjon. Selv om data publiseres med personidentifikasjon skal som regel øvrig data anonymiseres. Med anonymisering menes at datamaterialet bearbejdes slik at det ikke lenger er mulig å føre opplysningene tilbake til enkeltpersoner. Les mer om anonymisering .
13. Finansiering		
Hvordan finansieres prosjektet?	Ingen finansiering, kun masterprosjekt	
14. Tilleggsopplysninger		

Tilleggsopplysninger	<p>Puls fra webkamera registreres kun som et tall sammen med annen data. Ingen bilder/video vil bli lagret.</p> <p>Keystroke dynamics er en betegnelse på hvordan noen skriver på tastaturet (når en tast trykkes ned og slippes opp, og hvilken tast det gjelder). Dette er tidligere brukt til å gjenkjenne personer (biometrics) og detektere følelser. Systemet fungerer slik at brukeren får en programmeringsoppgave som brukeren løser i et nettleservindu. Etter at brukeren har valgt å kjøre koden blir han/hun spurt om å beskrive sin følelse (bored, confused, delighted, concentrated, frustrated, surprised) og hvilket nivå han/hun er på i programmering (beginner, intermediate, professional). Tastetrykk fra kodeskriving registreres sammen med de to andre attributtene. Det antas at hvordan noen bruker et tastatur varierer, slik som å skrive for hånd. Det blir derfor lagret en ID sammen med dataen fra den brukeren (f.eks.: bruker_id: 2). Navn/IP-adresse/andre personopplysninger blir ikke registrert eller lagret.</p>	
----------------------	---	--

Appendix C

Participant contract

Forespørsel om deltakelse i forskningsprosjektet

Detecting emotions during coding tutorials

Bakgrunn og formål

Masteroppgaven skal utforske muligheten for å registrere følelser til en bruker som gjør programmeringsoppgaver. Formålet er å gi en tilpasset progresjon og vanskelighetsgrad, og på sikt gi bedre læring, motivasjon og mestringsfølelse.

Forskningsspørsmål 1: Kan emosjoner bedre detekteres vha nøkkelord enn alle ord i koden?

Forskningsspørsmål 2: Kan deteksjon av puls forbedre klassifisering av følelser?

Prosjektet er en masteroppgave ved NTNU/IE/IDI. Veileder er Asbjørn Thomassen v/IDI. Resultatet av masteroppgaven vil være et proof of concept, der det blir studert om K nearest-neighbour er en god klassifiseringsmetode for følelser.

Deltakere til studien kontaktes direkte av masterstudenten.

Hva innebærer deltakelse i studien?

Ved å delta i studien vil du bli bedt om å gjennomføre programmeringsoppgaver i nettleseren. Mens du skriver kode vil alle tastetrykk registreres og lagres. Informasjonen som blir lagret er hvilken tast du trykket på, når den ble trykket ned, og når den ble sluppet opp igjen. Dette kalles keystroke dynamics. Etter at du har kjørt koden vil du bli spurt om hvilken følelse som best representerer din nåværende emosjonelle tilstand (kjedsomhet, forvirret, glad, konsentrert, frustrert eller overrasket), og på hvilket nivå innen programmering du er (nybegynner, viderekommen eller profesjonell).

Dataen du sender inn vil bli knyttet til deg ved et tilfeldig id-nummer, slik at det er mulig å finne igjen alle tastetrykkene for én bruker.

Du vil bli spurt om følgende for å kunne kategorisere deg:

- Hvilket kjønn er du (mann/kvinne)?
- Hvilken erfaring har du med programmering (fra 1 – 7)?
- Hvilke programmeringsspråk kjenner du til?
- Hvor mange år har du praktisert programmering totalt?
- Hvor mange år har du praktisk programmering før du startet på universitetet?
- Når lærte du først å programmere (dette er første gang, på egenhånd, før ungdomsskolen, ungdomsskolen, videregående, universitet/høgskole)?

Det vil også bli benyttet webkamera for å registrere pulsen til brukeren (du kan lese mer om hvordan her: <https://github.com/thearn/webcam-pulse-detector>). Ingen video/bilder blir lagret. Resultatet fra pulsmålingen er et tall som definerer pulsen din, som lagres sammen med dataen om tastetrykk.

Hva skjer med informasjonen om deg?

Alle personopplysninger vil bli behandlet konfidensielt. Kun masterstudenten (Thor Håkon Bredesen) vil ha tilgang til informasjonen som er registrert. Det vil ikke være mulig for en deltaker å kjenne igjen seg selv i den ferdige masteroppgaven. Resultatet vil kun bli presentert som hvor suksessfullt det var å bruke AI-metoden K nearest neighbour til å klassifisere følelser.

Prosjektet skal etter planen avsluttes 15. mai. Etter dette vil dataen bli anonymisert og lagret på ubestemt tid. Kun masterstudenten vil ha tilgang til dataen. Formålet med å fortsatt ha dataen er for å kunne videreutvikle produktet senere.

Frivillig deltakelse

Det er frivillig å delta i studien, og du kan når som helst trekke ditt samtykke uten å oppgi noen grunn. Dersom du trekker deg, vil alle opplysninger om deg bli anonymisert.

Dersom du ønsker å delta eller har spørsmål til studien, ta kontakt med masterstudent Thor Håkon Bredesen (976 29 575) eller veileder Asbjørn Thomassen (73 59 18 39).

Studien er meldt til Personvernombudet for forskning, NSD - Norsk senter for forskningsdata AS.

Samtykke til deltakelse i studien

Jeg har mottatt informasjon om studien, og er villig til å delta

(Signert av prosjektdeltaker, dato)

Appendix D

Response from Norwegian Centre for Research Data



Asbjørn Thomassen
Institutt for datateknologi og informatikk NTNU
Sem Sælandsvei 7-9
7491 TRONDHEIM

Vår dato: 03.04.2017

Vår ref: 53027 / 3 / AH

Deres dato:

Deres ref:

TILBAKEMELDING PÅ MELDING OM BEHANDLING AV PERSONOPPLYSNINGER

Vi viser til melding om behandling av personopplysninger, mottatt 15.02.2017. Meldingen gjelder prosjektet:

<i>53027</i>	<i>Detecting emotions during coding tutorials</i>
<i>Behandlingsansvarlig</i>	<i>NTNU, ved institusjonens øverste leder</i>
<i>Daglig ansvarlig</i>	<i>Asbjørn Thomassen</i>
<i>Student</i>	<i>Thor Håkon Bredesen</i>

Personvernombudet har vurdert prosjektet og finner at behandlingen av personopplysninger er meldepliktig i henhold til personopplysningsloven § 31. Behandlingen tilfredsstiller kravene i personopplysningsloven.

Personvernombudets vurdering forutsetter at prosjektet gjennomføres i tråd med opplysningene gitt i melde skjemaet, korrespondanse med ombudet, ombudets kommentarer samt personopplysningsloven og helseregisterloven med forskrifter. Behandlingen av personopplysninger kan settes i gang.

Det gjøres oppmerksom på at det skal gis ny melding dersom behandlingen endres i forhold til de opplysninger som ligger til grunn for personvernombudets vurdering. Endringsmeldinger gis via et eget skjema, http://www.nsd.uib.no/personvernombud/meld_prosjekt/meld_endringer.html. Det skal også gis melding etter tre år dersom prosjektet fortsatt pågår. Meldinger skal skje skriftlig til ombudet.

Personvernombudet har lagt ut opplysninger om prosjektet i en offentlig database, <http://pvo.nsd.no/prosjekt>.

Personvernombudet vil ved prosjektets avslutning, 15.05.2017, rette en henvendelse angående status for behandlingen av personopplysninger.

Vennlig hilsen

Kjersti Haugstvedt

Åsne Halskau

Kontaktperson: Åsne Halskau tlf: 55 58 21 88

Vedlegg: Prosjektvurdering

Dokumentet er elektronisk produsert og godkjent ved NSDs rutiner for elektronisk godkjenning.

Personvernombudet for forskning



Prosjektvurdering - Kommentar

Prosjektnr: 53027

Rekrutteringen skjer via eget nettverk. Ved rekruttering via eget nettverk er det spesielt viktig at forespørsel rettes på en slik måte at frivilligheten ved deltagelse ivaretas.

Utvalget informeres skriftlig om prosjektet og samtykker til deltakelse. Informasjonsskrivet er godt utformet, men personvernombudet anbefaler at det presiseres at det er et anonymt datasett som skal lagres videre etter prosjektslutt. Vi anbefaler også at tilleggsspørsmålene som skal stilles i spørreundersøkelsen tas inn i informasjonsskrivet, jf. tlf. samtale med student 03.04.2017.

Personvernombudet legger til grunn at forsker etterfølger NTNU sine interne rutiner for datasikkerhet. Dersom personopplysninger skal lagres på privat pc/mobile enheter, bør opplysningene krypteres tilstrekkelig.

Forventet prosjektslutt er 15.05.2017. Ifølge prosjektmeldingen skal innsamlede opplysninger da anonymiseres. Anonymisering innebærer å bearbeide datamaterialet slik at ingen enkeltpersoner kan gjenkjennes. Det gjøres ved å:

- slette direkte personopplysninger (som navn/koblingsnøkkel)
- slette/omskrive indirekte personopplysninger (identifiserende sammenstilling av bakgrunnsopplysninger som f.eks. bosted/arbeidssted, alder og kjønn)

Appendix E

Aggregated feature vectors by class

In these tables, the lowest and highest value for each feature, and the mean value, are presented. The values have not been normalised. The content of these tables have been used to create the graphs visible in figures 7.1, 7.2 (table E.1), 7.3 (table E.2), 7.4 (table E.3), 7.5 (table E.4), 7.6 (table E.5) and 7.7 (table E.6).

The feature’s name is found in the left most column, in the second column to the left are the keys used to derive the values. The keys are ordered by their occurrence, meaning that the first key is also the first in a digraph or trigraph. Mean, min and max values are expressed as milliseconds.

Table E.1: Mean, min and max values for aggregated bored feature vectors

Feature	Keys	Bored		
		<i>Mean</i>	<i>Min</i>	<i>Max</i>
number_of_misses		5.90	0.00	44.00
avg_dwell		151.86	71.00	568.68
avg_flight		45.58	-400.04	327.68
avg_keystrokes		333.36	129.42	580.36
first_key_down_second_key_down_time	c, o	161.86	16.65	1550.00
first_key_dwell_time	c	109.64	32.00	211.62
first_key_up_second_key_down_time	c, o	52.22	-122.60	1411.46
second_key_dwell_time	c, o	144.16	47.00	244.49
digraph_duration	c, o	306.02	127.90	1731.15
first_key_down_second_key_down_time	o, n	186.08	50.89	1026.05
first_key_dwell_time	o	144.16	47.00	244.49

Table E.1: Mean, min and max values for aggregated bored feature vectors

Feature	Keys	Bored		
		Mean	Min	Max
first_key_up_second_key_down_time	o, n	41.92	-127.97	961.38
second_key_dwell_time	o, n	117.04	47.00	197.16
digraph_duration	o, n	303.12	144.02	1112.47
first_key_down_second_key_down_time	n, s	236.09	40.03	2719.89
first_key_dwell_time	n	117.04	47.00	197.16
first_key_up_second_key_down_time	n, s	119.05	-93.19	2603.58
second_key_dwell_time	n, s	117.05	39.55	213.21
digraph_duration	n, s	353.13	112.01	2831.76
first_key_down_second_key_down_time	s, o	164.12	40.11	2063.88
first_key_dwell_time	s	117.05	39.55	213.21
first_key_up_second_key_down_time	s, o	47.07	-122.71	2024.32
second_key_dwell_time	s, o	97.82	47.00	159.35
digraph_duration	s, o	261.94	110.00	2190.46
first_key_down_second_key_down_time	o, l	186.98	96.14	297.00
first_key_dwell_time	o	97.82	47.00	159.35
first_key_up_second_key_down_time	o, l	89.15	15.92	219.00
second_key_dwell_time	o, l	107.80	32.00	200.81
digraph_duration	o, l	294.77	203.00	426.30
first_key_down_second_key_down_time	l, e	134.46	63.00	399.90
first_key_dwell_time	l	107.80	32.00	200.81
first_key_up_second_key_down_time	l, e	26.67	-58.62	311.91
second_key_dwell_time	l, e	110.46	47.99	212.75
digraph_duration	l, e	244.93	135.90	471.99
first_key_down_second_key_down_time	e, .	224.18	62.00	1184.49
first_key_dwell_time	e	110.46	47.99	212.75
first_key_up_second_key_down_time	e, .	113.72	-90.10	1068.36
second_key_dwell_time	e, .	107.84	47.00	164.72
digraph_duration	e, .	332.02	125.00	1267.63
first_key_down_second_key_down_time	., l	309.40	140.00	882.25
first_key_dwell_time	.	107.84	47.00	164.72
first_key_up_second_key_down_time	., l	201.56	40.23	792.00
second_key_dwell_time	., l	81.49	31.70	168.58
digraph_duration	., l	390.89	187.00	961.60
first_key_down_second_key_down_time	l, o	194.89	86.90	728.02
first_key_dwell_time	l	81.49	31.70	168.58
first_key_up_second_key_down_time	l, o	113.40	-23.71	696.02

Table E.1: Mean, min and max values for aggregated bored feature vectors

Feature	Keys	Bored		
		Mean	Min	Max
second_key_dwell_time	l, o	107.01	55.98	181.26
digraph_duration	l, o	301.89	174.27	856.14
first_key_down_second_key_down_time	o, g	168.31	58.60	1094.01
first_key_dwell_time	o	107.01	55.98	181.26
first_key_up_second_key_down_time	o, g	61.30	-95.94	1025.67
second_key_dwell_time	o, g	99.32	47.00	170.53
digraph_duration	o, g	267.62	143.83	1194.33
first_key_down_second_key_down_time	g, Shift	315.11	68.04	1000.09
first_key_dwell_time	g	99.32	47.00	170.53
first_key_up_second_key_down_time	g, Shift	215.79	-27.93	903.89
second_key_dwell_time	g, Shift	657.00	172.00	5631.10
digraph_duration	g, Shift	972.10	297.00	6040.21
first_key_down_second_key_down_time	Shift, (167.62	32.06	751.98
first_key_dwell_time	Shift	657.00	172.00	5631.10
first_key_up_second_key_down_time	Shift, (-489.38	-5350.87	-69.07
second_key_dwell_time	Shift, (117.51	48.00	197.16
digraph_duration	Shift, (285.13	120.08	871.97
first_key_down_second_key_down_time	c, o	161.86	16.65	1550.00
first_key_dwell_time	c	109.64	32.00	211.62
first_key_up_second_key_down_time	c, o	52.22	-122.60	1411.46
second_key_down_third_key_down_time	o, n	186.08	50.89	1026.05
second_key_dwell_time	o	144.16	47.00	244.49
second_key_up_third_key_down_time	o, n	41.92	-127.97	961.38
third_key_dwell_time	n	117.04	47.00	197.16
trigraph_duration	c, o, n	464.98	240.00	1784.55
first_key_down_second_key_down_time	o, n	186.08	50.89	1026.05
first_key_dwell_time	o	144.16	47.00	244.49
first_key_up_second_key_down_time	o, n	41.92	-127.97	961.38
second_key_down_third_key_down_time	n, s	236.09	40.03	2719.89
second_key_dwell_time	n	117.04	47.00	197.16
second_key_up_third_key_down_time	n, s	119.05	-93.19	2603.58
third_key_dwell_time	s	117.05	39.55	213.21
trigraph_duration	o, n, s	539.22	215.99	2996.68
first_key_down_second_key_down_time	n, s	236.09	40.03	2719.89
first_key_dwell_time	n	117.04	47.00	197.16
first_key_up_second_key_down_time	n, s	119.05	-93.19	2603.58

Table E.1: Mean, min and max values for aggregated bored feature vectors

Feature	Keys	Bored		
		Mean	Min	Max
second_key_down_third_key_down_time	s, o	164.12	40.11	2063.88
second_key_dwell_time	s	117.05	39.55	213.21
second_key_up_third_key_down_time	s, o	47.07	-122.71	2024.32
third_key_dwell_time	o	97.82	47.00	159.35
trigraph_duration	n, s, o	498.03	219.00	2944.21
first_key_down_second_key_down_time	s, o	164.12	40.11	2063.88
first_key_dwell_time	s	117.05	39.55	213.21
first_key_up_second_key_down_time	s, o	47.07	-122.71	2024.32
second_key_down_third_key_down_time	o, l	186.98	96.14	297.00
second_key_dwell_time	o	97.82	47.00	159.35
second_key_up_third_key_down_time	o, l	89.15	15.92	219.00
third_key_dwell_time	l	107.80	32.00	200.81
trigraph_duration	s, o, l	458.89	266.00	2360.95
first_key_down_second_key_down_time	o, l	186.98	96.14	297.00
first_key_dwell_time	o	97.82	47.00	159.35
first_key_up_second_key_down_time	o, l	89.15	15.92	219.00
second_key_down_third_key_down_time	l, e	134.46	63.00	399.90
second_key_dwell_time	l	107.80	32.00	200.81
second_key_up_third_key_down_time	l, e	26.67	-58.62	311.91
third_key_dwell_time	e	110.46	47.99	212.75
trigraph_duration	o, l, e	431.90	303.94	672.00
first_key_down_second_key_down_time	l, e	134.46	63.00	399.90
first_key_dwell_time	l	107.80	32.00	200.81
first_key_up_second_key_down_time	l, e	26.67	-58.62	311.91
second_key_down_third_key_down_time	e, .	224.18	62.00	1184.49
second_key_dwell_time	e	110.46	47.99	212.75
second_key_up_third_key_down_time	e, .	113.72	-90.10	1068.36
third_key_dwell_time	.	107.84	47.00	164.72
trigraph_duration	l, e, .	466.49	219.00	1351.25
first_key_down_second_key_down_time	e, .	224.18	62.00	1184.49
first_key_dwell_time	e	110.46	47.99	212.75
first_key_up_second_key_down_time	e, .	113.72	-90.10	1068.36
second_key_down_third_key_down_time	., l	309.40	140.00	882.25
second_key_dwell_time	.	107.84	47.00	164.72
second_key_up_third_key_down_time	., l	201.56	40.23	792.00
third_key_dwell_time	l	81.49	31.70	168.58

Table E.1: Mean, min and max values for aggregated bored feature vectors

Feature	Keys	Bored		
		Mean	Min	Max
trigraph_duration	e, ., l	615.07	265.00	1444.04
first_key_down_second_key_down_time	., l	309.40	140.00	882.25
first_key_dwell_time	.	107.84	47.00	164.72
first_key_up_second_key_down_time	., l	201.56	40.23	792.00
second_key_down_third_key_down_time	l, o	194.89	86.90	728.02
second_key_dwell_time	l	81.49	31.70	168.58
second_key_up_third_key_down_time	l, o	113.40	-23.71	696.02
third_key_dwell_time	o	107.01	55.98	181.26
trigraph_duration	., l, o	611.29	343.00	1319.96
first_key_down_second_key_down_time	l, o	194.89	86.90	728.02
first_key_dwell_time	l	81.49	31.70	168.58
first_key_up_second_key_down_time	l, o	113.40	-23.71	696.02
second_key_down_third_key_down_time	o, g	168.31	58.60	1094.01
second_key_dwell_time	o	107.01	55.98	181.26
second_key_up_third_key_down_time	o, g	61.30	-95.94	1025.67
third_key_dwell_time	g	99.32	47.00	170.53
trigraph_duration	l, o, g	462.51	287.97	1402.40
first_key_down_second_key_down_time	o, g	168.31	58.60	1094.01
first_key_dwell_time	o	107.01	55.98	181.26
first_key_up_second_key_down_time	o, g	61.30	-95.94	1025.67
second_key_down_third_key_down_time	g, Shift	315.11	68.04	1000.09
second_key_dwell_time	g	99.32	47.00	170.53
second_key_up_third_key_down_time	g, Shift	215.79	-27.93	903.89
third_key_dwell_time	Shift	657.00	172.00	5631.10
trigraph_duration	o, g, Shift	1140.41	406.00	6130.87
first_key_down_second_key_down_time	g, Shift	315.11	68.04	1000.09
first_key_dwell_time	g	99.32	47.00	170.53
first_key_up_second_key_down_time	g, Shift	215.79	-27.93	903.89
second_key_down_third_key_down_time	Shift, (167.62	32.06	751.98
second_key_dwell_time	Shift	657.00	172.00	5631.10
second_key_up_third_key_down_time	Shift, (-489.38	-5350.87	-69.07
third_key_dwell_time	(117.51	48.00	197.16
trigraph_duration	g, Shift, (600.24	234.00	1176.04

Table E.2: Mean, min and max values for aggregated concentrated feature vectors

Feature	Keys	Concentrated		
		Mean	Min	Max
number_of_misses		7.51	0.00	88.00
avg_dwell		128.48	71.92	341.48
avg_flight		69.45	-199.13	797.61
avg_keystrokes		355.75	65.13	665.53
first_key_down_second_key_down_time	c, o	141.27	15.93	1610.85
first_key_dwell_time	c	102.61	13.71	226.94
first_key_up_second_key_down_time	c, o	38.66	-117.16	1477.91
second_key_dwell_time	c, o	121.58	39.88	364.76
digraph_duration	c, o	262.85	87.86	1823.86
first_key_down_second_key_down_time	o, n	143.37	19.12	2719.34
first_key_dwell_time	o	121.58	39.88	364.76
first_key_up_second_key_down_time	o, n	21.79	-175.01	2628.34
second_key_dwell_time	o, n	104.47	48.02	208.67
digraph_duration	o, n	247.83	112.01	2815.64
first_key_down_second_key_down_time	n, s	190.37	23.91	3006.76
first_key_dwell_time	n	104.47	48.02	208.67
first_key_up_second_key_down_time	n, s	85.90	-130.88	2905.28
second_key_dwell_time	n, s	106.74	46.33	202.43
digraph_duration	n, s	297.12	119.72	3114.60
first_key_down_second_key_down_time	s, o	124.64	31.89	1707.31
first_key_dwell_time	s	106.74	46.33	202.43
first_key_up_second_key_down_time	s, o	17.89	-106.42	1559.20
second_key_dwell_time	s, o	90.24	32.70	163.89
digraph_duration	s, o	214.87	74.34	1805.53
first_key_down_second_key_down_time	o, l	209.70	103.91	1928.64
first_key_dwell_time	o	90.24	32.70	163.89
first_key_up_second_key_down_time	o, l	119.46	7.92	1821.80
second_key_dwell_time	o, l	96.94	40.00	176.03
digraph_duration	o, l	306.63	184.09	2026.35
first_key_down_second_key_down_time	l, e	170.02	32.05	1928.21
first_key_dwell_time	l	96.94	40.00	176.03
first_key_up_second_key_down_time	l, e	73.09	-72.06	1774.73
second_key_dwell_time	l, e	97.31	39.30	207.90
digraph_duration	l, e	267.33	104.10	2104.38
first_key_down_second_key_down_time	e, .	365.52	47.98	9712.00
first_key_dwell_time	e	97.31	39.30	207.90

Table E.2: Mean, min and max values for aggregated concentrated feature vectors

Feature	Keys	Concentrated		
		Mean	Min	Max
first_key_up_second_key_down_time	e, .	268.21	-90.32	9608.05
second_key_dwell_time	e, .	91.88	31.00	173.06
digraph_duration	e, .	457.40	127.94	9807.96
first_key_down_second_key_down_time	., l	289.12	125.00	2453.36
first_key_dwell_time	.	91.88	31.00	173.06
first_key_up_second_key_down_time	., l	197.24	44.23	2320.76
second_key_dwell_time	., l	84.22	10.31	193.77
digraph_duration	., l	373.34	188.00	2518.72
first_key_down_second_key_down_time	l, o	181.51	64.13	840.02
first_key_dwell_time	l	84.22	10.31	193.77
first_key_up_second_key_down_time	l, o	97.30	-65.48	803.74
second_key_dwell_time	l, o	94.53	24.01	181.20
digraph_duration	l, o	276.04	128.13	912.00
first_key_down_second_key_down_time	o, g	126.38	23.74	1176.03
first_key_dwell_time	o	94.53	24.01	181.20
first_key_up_second_key_down_time	o, g	31.85	-102.54	1072.03
second_key_dwell_time	o, g	93.82	28.82	179.30
digraph_duration	o, g	220.20	113.08	1263.89
first_key_down_second_key_down_time	g, Shift	369.71	85.33	8479.75
first_key_dwell_time	g	93.82	28.82	179.30
first_key_up_second_key_down_time	g, Shift	275.89	-42.59	8359.74
second_key_dwell_time	g, Shift	482.18	149.99	2898.16
digraph_duration	g, Shift	851.89	250.00	8679.85
first_key_down_second_key_down_time	Shift, (157.74	0.00	1015.00
first_key_dwell_time	Shift	482.18	149.99	2898.16
first_key_up_second_key_down_time	Shift, (-324.44	-2771.33	-37.80
second_key_dwell_time	Shift, (103.78	44.10	216.01
digraph_duration	Shift, (261.52	63.83	1078.00
first_key_down_second_key_down_time	c, o	141.27	15.93	1610.85
first_key_dwell_time	c	102.61	13.71	226.94
first_key_up_second_key_down_time	c, o	38.66	-117.16	1477.91
second_key_down_third_key_down_time	o, n	143.37	19.12	2719.34
second_key_dwell_time	o	121.58	39.88	364.76
second_key_up_third_key_down_time	o, n	21.79	-175.01	2628.34
third_key_dwell_time	n	104.47	48.02	208.67
trigraph_duration	c, o, n	389.11	196.56	2931.40

Table E.2: Mean, min and max values for aggregated concentrated feature vectors

Feature	Keys	Concentrated		
		Mean	Min	Max
first_key_down_second_key_down_time	o, n	143.37	19.12	2719.34
first_key_dwell_time	o	121.58	39.88	364.76
first_key_up_second_key_down_time	o, n	21.79	-175.01	2628.34
second_key_down_third_key_down_time	n, s	190.37	23.91	3006.76
second_key_dwell_time	n	104.47	48.02	208.67
second_key_up_third_key_down_time	n, s	85.90	-130.88	2905.28
third_key_dwell_time	s	106.74	46.33	202.43
trigraph_duration	o, n, s	440.48	183.80	3332.88
first_key_down_second_key_down_time	n, s	190.37	23.91	3006.76
first_key_dwell_time	n	104.47	48.02	208.67
first_key_up_second_key_down_time	n, s	85.90	-130.88	2905.28
second_key_down_third_key_down_time	s, o	124.64	31.89	1707.31
second_key_dwell_time	s	106.74	46.33	202.43
second_key_up_third_key_down_time	s, o	17.89	-106.42	1559.20
third_key_dwell_time	o	90.24	32.70	163.89
trigraph_duration	n, s, o	405.25	177.32	4035.71
first_key_down_second_key_down_time	s, o	124.64	31.89	1707.31
first_key_dwell_time	s	106.74	46.33	202.43
first_key_up_second_key_down_time	s, o	17.89	-106.42	1559.20
second_key_down_third_key_down_time	o, l	209.70	103.91	1928.64
second_key_dwell_time	o	90.24	32.70	163.89
second_key_up_third_key_down_time	o, l	119.46	7.92	1821.80
third_key_dwell_time	l	96.94	40.00	176.03
trigraph_duration	s, o, l	431.27	239.81	2094.71
first_key_down_second_key_down_time	o, l	209.70	103.91	1928.64
first_key_dwell_time	o	90.24	32.70	163.89
first_key_up_second_key_down_time	o, l	119.46	7.92	1821.80
second_key_down_third_key_down_time	l, e	170.02	32.05	1928.21
second_key_dwell_time	l	96.94	40.00	176.03
second_key_up_third_key_down_time	l, e	73.09	-72.06	1774.73
third_key_dwell_time	e	97.31	39.30	207.90
trigraph_duration	o, l, e	477.03	272.01	2296.29
first_key_down_second_key_down_time	l, e	170.02	32.05	1928.21
first_key_dwell_time	l	96.94	40.00	176.03
first_key_up_second_key_down_time	l, e	73.09	-72.06	1774.73
second_key_down_third_key_down_time	e, .	365.52	47.98	9712.00

Table E.2: Mean, min and max values for aggregated concentrated feature vectors

Feature	Keys	Concentrated		
		Mean	Min	Max
second_key_dwell_time	e	97.31	39.30	207.90
second_key_up_third_key_down_time	e, .	268.21	-90.32	9608.05
third_key_dwell_time	.	91.88	31.00	173.06
trigraph_duration	l, e, .	627.42	218.00	9919.85
first_key_down_second_key_down_time	e, .	365.52	47.98	9712.00
first_key_dwell_time	e	97.31	39.30	207.90
first_key_up_second_key_down_time	e, .	268.21	-90.32	9608.05
second_key_down_third_key_down_time	., l	289.12	125.00	2453.36
second_key_dwell_time	.	91.88	31.00	173.06
second_key_up_third_key_down_time	., l	197.24	44.23	2320.76
third_key_dwell_time	l	84.22	10.31	193.77
trigraph_duration	e, ., l	738.86	272.36	10039.93
first_key_down_second_key_down_time	., l	289.12	125.00	2453.36
first_key_dwell_time	.	91.88	31.00	173.06
first_key_up_second_key_down_time	., l	197.24	44.23	2320.76
second_key_down_third_key_down_time	l, o	181.51	64.13	840.02
second_key_dwell_time	l	84.22	10.31	193.77
second_key_up_third_key_down_time	l, o	97.30	-65.48	803.74
third_key_dwell_time	o	94.53	24.01	181.20
trigraph_duration	., l, o	565.17	285.67	2758.57
first_key_down_second_key_down_time	l, o	181.51	64.13	840.02
first_key_dwell_time	l	84.22	10.31	193.77
first_key_up_second_key_down_time	l, o	97.30	-65.48	803.74
second_key_down_third_key_down_time	o, g	126.38	23.74	1176.03
second_key_dwell_time	o	94.53	24.01	181.20
second_key_up_third_key_down_time	o, g	31.85	-102.54	1072.03
third_key_dwell_time	g	93.82	28.82	179.30
trigraph_duration	l, o, g	401.71	232.62	1391.84
first_key_down_second_key_down_time	o, g	126.38	23.74	1176.03
first_key_dwell_time	o	94.53	24.01	181.20
first_key_up_second_key_down_time	o, g	31.85	-102.54	1072.03
second_key_down_third_key_down_time	g, Shift	369.71	85.33	8479.75
second_key_dwell_time	g	93.82	28.82	179.30
second_key_up_third_key_down_time	g, Shift	275.89	-42.59	8359.74
third_key_dwell_time	Shift	482.18	149.99	2898.16
trigraph_duration	o, g, Shift	978.27	306.98	8759.84

Table E.2: Mean, min and max values for aggregated concentrated feature vectors

Feature	Keys	Concentrated		
		Mean	Min	Max
first_key_down_second_key_down_time	g, Shift	369.71	85.33	8479.75
first_key_dwell_time	g	93.82	28.82	179.30
first_key_up_second_key_down_time	g, Shift	275.89	-42.59	8359.74
second_key_down_third_key_down_time	Shift, (157.74	0.00	1015.00
second_key_dwell_time	Shift	482.18	149.99	2898.16
second_key_up_third_key_down_time	Shift, (-324.44	-2771.33	-37.80
third_key_dwell_time	(103.78	44.10	216.01
trigraph_duration	g, Shift, (631.24	234.00	8655.84

Table E.3: Mean, min and max values for aggregated confused feature vectors

Feature	Keys	Confused		
		Mean	Min	Max
number_of_misses		9.48	0.00	105.00
avg_dwell		124.08	77.08	222.58
avg_flight		262.06	-17.86	4806.74
avg_keystrokes		357.64	12.20	557.11
first_key_down_second_key_down_time	c, o	121.15	5.04	484.00
first_key_dwell_time	c	102.19	63.00	145.82
first_key_up_second_key_down_time	c, o	18.97	-122.65	375.00
second_key_dwell_time	c, o	114.78	47.00	203.00
digraph_duration	c, o	235.94	112.00	687.00
first_key_down_second_key_down_time	o, n	134.37	47.99	877.75
first_key_dwell_time	o	114.78	47.00	203.00
first_key_up_second_key_down_time	o, n	19.58	-79.99	785.29
second_key_dwell_time	o, n	107.31	47.00	171.00
digraph_duration	o, n	241.68	148.02	952.26
first_key_down_second_key_down_time	n, s	164.99	31.75	1156.00
first_key_dwell_time	n	107.31	47.00	171.00
first_key_up_second_key_down_time	n, s	57.68	-71.86	985.00
second_key_dwell_time	n, s	97.28	70.62	143.88
digraph_duration	n, s	262.27	119.78	1265.00
first_key_down_second_key_down_time	s, o	112.02	44.13	204.00
first_key_dwell_time	s	97.28	70.62	143.88
first_key_up_second_key_down_time	s, o	14.74	-86.05	94.00
second_key_dwell_time	s, o	84.03	55.99	143.89

Table E.3: Mean, min and max values for aggregated confused feature vectors

Feature	Keys	Confused		
		Mean	Min	Max
digraph_duration	s, o	196.06	131.08	344.00
first_key_down_second_key_down_time	o, l	178.83	79.99	272.21
first_key_dwell_time	o	84.03	55.99	143.89
first_key_up_second_key_down_time	o, l	94.80	16.08	184.10
second_key_dwell_time	o, l	96.81	41.18	144.01
digraph_duration	o, l	275.64	199.97	407.96
first_key_down_second_key_down_time	l, e	282.75	80.02	3576.61
first_key_dwell_time	l	96.81	41.18	144.01
first_key_up_second_key_down_time	l, e	185.94	-40.07	3470.02
second_key_dwell_time	l, e	94.92	64.03	197.20
digraph_duration	l, e	377.66	160.01	3773.81
first_key_down_second_key_down_time	e, .	374.11	78.00	2471.95
first_key_dwell_time	e	94.92	64.03	197.20
first_key_up_second_key_down_time	e, .	279.19	-8.02	2384.00
second_key_dwell_time	e, .	91.03	61.37	154.56
digraph_duration	e, .	465.14	141.00	2543.95
first_key_down_second_key_down_time	., l	249.13	156.00	807.83
first_key_dwell_time	.	91.03	61.37	154.56
first_key_up_second_key_down_time	., l	158.09	70.25	727.99
second_key_dwell_time	., l	81.69	40.06	150.99
digraph_duration	., l	330.81	219.00	847.89
first_key_down_second_key_down_time	l, o	171.96	105.50	235.00
first_key_dwell_time	l	81.69	40.06	150.99
first_key_up_second_key_down_time	l, o	90.27	-45.49	151.96
second_key_dwell_time	l, o	90.88	64.06	164.93
digraph_duration	l, o	262.84	173.26	385.12
first_key_down_second_key_down_time	o, g	105.46	64.01	187.83
first_key_dwell_time	o	90.88	64.06	164.93
first_key_up_second_key_down_time	o, g	14.58	-69.23	105.32
second_key_dwell_time	o, g	90.38	56.11	191.85
digraph_duration	o, g	195.84	127.99	287.55
first_key_down_second_key_down_time	g, Shift	2833.62	80.15	62399.42
first_key_dwell_time	g	90.38	56.11	191.85
first_key_up_second_key_down_time	g, Shift	2743.25	15.95	62319.50
second_key_dwell_time	g, Shift	467.78	191.73	1188.00
digraph_duration	g, Shift	3301.40	271.88	62831.62

Table E.3: Mean, min and max values for aggregated confused feature vectors

Feature	Keys	Confused		
		Mean	Min	Max
first_key_down_second_key_down_time	Shift, (197.54	23.79	543.74
first_key_dwell_time	Shift	467.78	191.73	1188.00
first_key_up_second_key_down_time	Shift, (-270.24	-797.00	-55.68
second_key_dwell_time	Shift, (93.94	40.02	192.01
digraph_duration	Shift, (291.48	119.77	703.91
first_key_down_second_key_down_time	c, o	121.15	5.04	484.00
first_key_dwell_time	c	102.19	63.00	145.82
first_key_up_second_key_down_time	c, o	18.97	-122.65	375.00
second_key_down_third_key_down_time	o, n	134.37	47.99	877.75
second_key_dwell_time	o	114.78	47.00	203.00
second_key_up_third_key_down_time	o, n	19.58	-79.99	785.29
third_key_dwell_time	n	107.31	47.00	171.00
trigraph_duration	c, o, n	362.83	240.00	1079.22
first_key_down_second_key_down_time	o, n	134.37	47.99	877.75
first_key_dwell_time	o	114.78	47.00	203.00
first_key_up_second_key_down_time	o, n	19.58	-79.99	785.29
second_key_down_third_key_down_time	n, s	164.99	31.75	1156.00
second_key_dwell_time	n	107.31	47.00	171.00
second_key_up_third_key_down_time	n, s	57.68	-71.86	985.00
third_key_dwell_time	s	97.28	70.62	143.88
trigraph_duration	o, n, s	396.64	224.20	1406.00
first_key_down_second_key_down_time	n, s	164.99	31.75	1156.00
first_key_dwell_time	n	107.31	47.00	171.00
first_key_up_second_key_down_time	n, s	57.68	-71.86	985.00
second_key_down_third_key_down_time	s, o	112.02	44.13	204.00
second_key_dwell_time	s	97.28	70.62	143.88
second_key_up_third_key_down_time	s, o	14.74	-86.05	94.00
third_key_dwell_time	o	84.03	55.99	143.89
trigraph_duration	n, s, o	361.04	234.00	1390.00
first_key_down_second_key_down_time	s, o	112.02	44.13	204.00
first_key_dwell_time	s	97.28	70.62	143.88
first_key_up_second_key_down_time	s, o	14.74	-86.05	94.00
second_key_down_third_key_down_time	o, l	178.83	79.99	272.21
second_key_dwell_time	o	84.03	55.99	143.89
second_key_up_third_key_down_time	o, l	94.80	16.08	184.10
third_key_dwell_time	l	96.81	41.18	144.01

Table E.3: Mean, min and max values for aggregated confused feature vectors

Feature	Keys	Confused		
		Mean	Min	Max
trigraph_duration	s, o, l	387.66	300.54	567.83
first_key_down_second_key_down_time	o, l	178.83	79.99	272.21
first_key_dwell_time	o	84.03	55.99	143.89
first_key_up_second_key_down_time	o, l	94.80	16.08	184.10
second_key_down_third_key_down_time	l, e	282.75	80.02	3576.61
second_key_dwell_time	l	96.81	41.18	144.01
second_key_up_third_key_down_time	l, e	185.94	-40.07	3470.02
third_key_dwell_time	e	94.92	64.03	197.20
trigraph_duration	o, l, e	556.50	255.99	4020.02
first_key_down_second_key_down_time	l, e	282.75	80.02	3576.61
first_key_dwell_time	l	96.81	41.18	144.01
first_key_up_second_key_down_time	l, e	185.94	-40.07	3470.02
second_key_down_third_key_down_time	e, .	374.11	78.00	2471.95
second_key_dwell_time	e	94.92	64.03	197.20
second_key_up_third_key_down_time	e, .	279.19	-8.02	2384.00
third_key_dwell_time	.	91.03	61.37	154.56
trigraph_duration	l, e, .	747.89	250.00	4170.29
first_key_down_second_key_down_time	e, .	374.11	78.00	2471.95
first_key_dwell_time	e	94.92	64.03	197.20
first_key_up_second_key_down_time	e, .	279.19	-8.02	2384.00
second_key_down_third_key_down_time	., l	249.13	156.00	807.83
second_key_dwell_time	.	91.03	61.37	154.56
second_key_up_third_key_down_time	., l	158.09	70.25	727.99
third_key_dwell_time	l	81.69	40.06	150.99
trigraph_duration	e, ., l	704.92	297.00	2776.01
first_key_down_second_key_down_time	., l	249.13	156.00	807.83
first_key_dwell_time	.	91.03	61.37	154.56
first_key_up_second_key_down_time	., l	158.09	70.25	727.99
second_key_down_third_key_down_time	l, o	171.96	105.50	235.00
second_key_dwell_time	l	81.69	40.06	150.99
second_key_up_third_key_down_time	l, o	90.27	-45.49	151.96
third_key_dwell_time	o	90.88	64.06	164.93
trigraph_duration	., l, o	511.97	391.00	1111.98
first_key_down_second_key_down_time	l, o	171.96	105.50	235.00
first_key_dwell_time	l	81.69	40.06	150.99
first_key_up_second_key_down_time	l, o	90.27	-45.49	151.96

Table E.3: Mean, min and max values for aggregated confused feature vectors

Feature	Keys	Confused		
		Mean	Min	Max
second_key_down_third_key_down_time	o, g	105.46	64.01	187.83
second_key_dwell_time	o	90.88	64.06	164.93
second_key_up_third_key_down_time	o, g	14.58	-69.23	105.32
third_key_dwell_time	g	90.38	56.11	191.85
trigraph_duration	l, o, g	367.80	244.57	507.73
first_key_down_second_key_down_time	o, g	105.46	64.01	187.83
first_key_dwell_time	o	90.88	64.06	164.93
first_key_up_second_key_down_time	o, g	14.58	-69.23	105.32
second_key_down_third_key_down_time	g, Shift	2833.62	80.15	62399.42
second_key_dwell_time	g	90.38	56.11	191.85
second_key_up_third_key_down_time	g, Shift	2743.25	15.95	62319.50
third_key_dwell_time	Shift	467.78	191.73	1188.00
trigraph_duration	o, g, Shift	3406.87	383.66	62935.71
first_key_down_second_key_down_time	g, Shift	2833.62	80.15	62399.42
first_key_dwell_time	g	90.38	56.11	191.85
first_key_up_second_key_down_time	g, Shift	2743.25	15.95	62319.50
second_key_down_third_key_down_time	Shift, (197.54	23.79	543.74
second_key_dwell_time	Shift	467.78	191.73	1188.00
second_key_up_third_key_down_time	Shift, (-270.24	-797.00	-55.68
third_key_dwell_time	(93.94	40.02	192.01
trigraph_duration	g, Shift, (3125.11	263.92	62711.18

Table E.4: Mean, min and max values for aggregated delighted feature vectors

Feature	Keys	Delighted		
		Mean	Min	Max
number_of_misses		3.20	0.00	29.00
avg_dwell		119.63	72.62	339.08
avg_flight		40.56	-155.40	315.40
avg_keystrokes		410.56	130.44	548.52
first_key_down_second_key_down_time	c, o	108.75	24.01	783.85
first_key_dwell_time	c	97.72	48.18	175.89
first_key_up_second_key_down_time	c, o	11.04	-85.33	639.87
second_key_dwell_time	c, o	118.16	31.00	227.90
digraph_duration	c, o	226.91	88.00	919.85
first_key_down_second_key_down_time	o, n	110.61	31.82	529.50

Table E.4: Mean, min and max values for aggregated delighted feature vectors

Feature	Keys	Delighted		
		Mean	Min	Max
first_key_dwell_time	o	118.16	31.00	227.90
first_key_up_second_key_down_time	o, n	-7.55	-143.10	438.52
second_key_dwell_time	o, n	104.96	47.00	169.73
digraph_duration	o, n	215.57	120.00	611.33
first_key_down_second_key_down_time	n, s	113.80	15.98	326.54
first_key_dwell_time	n	104.96	47.00	169.73
first_key_up_second_key_down_time	n, s	8.84	-63.99	244.54
second_key_dwell_time	n, s	94.84	45.26	180.26
digraph_duration	n, s	208.64	111.98	405.55
first_key_down_second_key_down_time	s, o	118.67	32.16	935.98
first_key_dwell_time	s	94.84	45.26	180.26
first_key_up_second_key_down_time	s, o	23.84	-106.55	839.99
second_key_dwell_time	s, o	84.25	46.00	127.89
digraph_duration	s, o	202.92	122.62	1023.97
first_key_down_second_key_down_time	o, l	193.62	95.99	1274.48
first_key_dwell_time	o	84.25	46.00	127.89
first_key_up_second_key_down_time	o, l	109.37	8.01	1167.94
second_key_dwell_time	o, l	92.12	31.95	183.98
digraph_duration	o, l	285.74	203.00	1365.44
first_key_down_second_key_down_time	l, e	152.84	56.08	2272.10
first_key_dwell_time	l	92.12	31.95	183.98
first_key_up_second_key_down_time	l, e	60.72	-80.02	2160.00
second_key_dwell_time	l, e	87.24	48.01	174.84
digraph_duration	l, e	240.08	144.04	2344.07
first_key_down_second_key_down_time	e, .	216.18	63.00	1626.79
first_key_dwell_time	e	87.24	48.01	174.84
first_key_up_second_key_down_time	e, .	128.94	-68.42	1548.05
second_key_dwell_time	e, .	82.44	31.00	157.95
digraph_duration	e, .	298.62	109.00	1706.35
first_key_down_second_key_down_time	., l	238.53	140.00	981.98
first_key_dwell_time	.	82.44	31.00	157.95
first_key_up_second_key_down_time	., l	156.09	46.34	865.48
second_key_dwell_time	., l	70.67	31.81	152.00
digraph_duration	., l	309.20	187.00	1040.20
first_key_down_second_key_down_time	l, o	186.06	87.03	1040.00
first_key_dwell_time	l	70.67	31.81	152.00

Table E.4: Mean, min and max values for aggregated delighted feature vectors

Feature	Keys	Delighted		
		Mean	Min	Max
first_key_up_second_key_down_time	l, o	115.39	-62.97	960.15
second_key_dwell_time	l, o	84.70	32.05	165.42
digraph_duration	l, o	270.76	170.89	1127.85
first_key_down_second_key_down_time	o, g	137.66	31.65	1232.02
first_key_dwell_time	o	84.70	32.05	165.42
first_key_up_second_key_down_time	o, g	52.96	-79.87	1151.89
second_key_dwell_time	o, g	81.60	46.00	135.98
digraph_duration	o, g	219.26	136.00	1352.04
first_key_down_second_key_down_time	g, Shift	216.04	88.10	1456.00
first_key_dwell_time	g	81.60	46.00	135.98
first_key_up_second_key_down_time	g, Shift	134.45	0.85	1376.00
second_key_dwell_time	g, Shift	461.29	136.68	2702.68
digraph_duration	g, Shift	677.33	250.00	2948.62
first_key_down_second_key_down_time	Shift, (194.47	26.52	2359.94
first_key_dwell_time	Shift	461.29	136.68	2702.68
first_key_up_second_key_down_time	Shift, (-266.82	-2676.16	-40.16
second_key_dwell_time	Shift, (95.26	31.00	192.09
digraph_duration	Shift, (289.73	94.00	2511.97
first_key_down_second_key_down_time	c, o	108.75	24.01	783.85
first_key_dwell_time	c	97.72	48.18	175.89
first_key_up_second_key_down_time	c, o	11.04	-85.33	639.87
second_key_down_third_key_down_time	o, n	110.61	31.82	529.50
second_key_dwell_time	o	118.16	31.00	227.90
second_key_up_third_key_down_time	o, n	-7.55	-143.10	438.52
third_key_dwell_time	n	104.96	47.00	169.73
trigraph_duration	c, o, n	324.32	224.00	1167.91
first_key_down_second_key_down_time	o, n	110.61	31.82	529.50
first_key_dwell_time	o	118.16	31.00	227.90
first_key_up_second_key_down_time	o, n	-7.55	-143.10	438.52
second_key_down_third_key_down_time	n, s	113.80	15.98	326.54
second_key_dwell_time	n	104.96	47.00	169.73
second_key_up_third_key_down_time	n, s	8.84	-63.99	244.54
third_key_dwell_time	s	94.84	45.26	180.26
trigraph_duration	o, n, s	319.25	192.00	935.05
first_key_down_second_key_down_time	n, s	113.80	15.98	326.54
first_key_dwell_time	n	104.96	47.00	169.73

Table E.4: Mean, min and max values for aggregated delighted feature vectors

Feature	Keys	Delighted		
		Mean	Min	Max
first_key_up_second_key_down_time	n, s	8.84	-63.99	244.54
second_key_down_third_key_down_time	s, o	118.67	32.16	935.98
second_key_dwell_time	s	94.84	45.26	180.26
second_key_up_third_key_down_time	s, o	23.84	-106.55	839.99
third_key_dwell_time	o	84.25	46.00	127.89
trigraph_duration	n, s, o	316.73	234.00	1047.98
first_key_down_second_key_down_time	s, o	118.67	32.16	935.98
first_key_dwell_time	s	94.84	45.26	180.26
first_key_up_second_key_down_time	s, o	23.84	-106.55	839.99
second_key_down_third_key_down_time	o, l	193.62	95.99	1274.48
second_key_dwell_time	o	84.25	46.00	127.89
second_key_up_third_key_down_time	o, l	109.37	8.01	1167.94
third_key_dwell_time	l	92.12	31.95	183.98
trigraph_duration	s, o, l	404.41	281.00	1509.25
first_key_down_second_key_down_time	o, l	193.62	95.99	1274.48
first_key_dwell_time	o	84.25	46.00	127.89
first_key_up_second_key_down_time	o, l	109.37	8.01	1167.94
second_key_down_third_key_down_time	l, e	152.84	56.08	2272.10
second_key_dwell_time	l	92.12	31.95	183.98
second_key_up_third_key_down_time	l, e	60.72	-80.02	2160.00
third_key_dwell_time	e	87.24	48.01	174.84
trigraph_duration	o, l, e	433.70	272.01	2503.98
first_key_down_second_key_down_time	l, e	152.84	56.08	2272.10
first_key_dwell_time	l	92.12	31.95	183.98
first_key_up_second_key_down_time	l, e	60.72	-80.02	2160.00
second_key_down_third_key_down_time	e, .	216.18	63.00	1626.79
second_key_dwell_time	e	87.24	48.01	174.84
second_key_up_third_key_down_time	e, .	128.94	-68.42	1548.05
third_key_dwell_time	.	82.44	31.00	157.95
trigraph_duration	l, e, .	451.46	219.00	2488.09
first_key_down_second_key_down_time	e, .	216.18	63.00	1626.79
first_key_dwell_time	e	87.24	48.01	174.84
first_key_up_second_key_down_time	e, .	128.94	-68.42	1548.05
second_key_down_third_key_down_time	., l	238.53	140.00	981.98
second_key_dwell_time	.	82.44	31.00	157.95
second_key_up_third_key_down_time	., l	156.09	46.34	865.48

Table E.4: Mean, min and max values for aggregated delighted feature vectors

Feature	Keys	Delighted		
		Mean	Min	Max
third_key_dwell_time	l	70.67	31.81	152.00
trigraph_duration	e, ., l	525.37	265.00	1960.85
first_key_down_second_key_down_time	., l	238.53	140.00	981.98
first_key_dwell_time	.	82.44	31.00	157.95
first_key_up_second_key_down_time	., l	156.09	46.34	865.48
second_key_down_third_key_down_time	l, o	186.06	87.03	1040.00
second_key_dwell_time	l	70.67	31.81	152.00
second_key_up_third_key_down_time	l, o	115.39	-62.97	960.15
third_key_dwell_time	o	84.70	32.05	165.42
trigraph_duration	., l, o	509.29	343.00	1416.04
first_key_down_second_key_down_time	l, o	186.06	87.03	1040.00
first_key_dwell_time	l	70.67	31.81	152.00
first_key_up_second_key_down_time	l, o	115.39	-62.97	960.15
second_key_down_third_key_down_time	o, g	137.66	31.65	1232.02
second_key_dwell_time	o	84.70	32.05	165.42
second_key_up_third_key_down_time	o, g	52.96	-79.87	1151.89
third_key_dwell_time	g	81.60	46.00	135.98
trigraph_duration	l, o, g	405.32	289.07	1511.93
first_key_down_second_key_down_time	o, g	137.66	31.65	1232.02
first_key_dwell_time	o	84.70	32.05	165.42
first_key_up_second_key_down_time	o, g	52.96	-79.87	1151.89
second_key_down_third_key_down_time	g, Shift	216.04	88.10	1456.00
second_key_dwell_time	g	81.60	46.00	135.98
second_key_up_third_key_down_time	g, Shift	134.45	0.85	1376.00
third_key_dwell_time	Shift	461.29	136.68	2702.68
trigraph_duration	o, g, Shift	814.99	344.00	3039.23
first_key_down_second_key_down_time	g, Shift	216.04	88.10	1456.00
first_key_dwell_time	g	81.60	46.00	135.98
first_key_up_second_key_down_time	g, Shift	134.45	0.85	1376.00
second_key_down_third_key_down_time	Shift, (194.47	26.52	2359.94
second_key_dwell_time	Shift	461.29	136.68	2702.68
second_key_up_third_key_down_time	Shift, (-266.82	-2676.16	-40.16
third_key_dwell_time	(95.26	31.00	192.09
trigraph_duration	g, Shift, (505.77	219.00	2784.23

Table E.5: Mean, min and max values for aggregated frustrated feature vectors

Feature	Keys	Frustrated		
		Mean	Min	Max
number_of_misses		16.79	0.00	81.00
avg_dwell		129.24	75.77	269.54
avg_flight		44.84	-108.31	609.46
avg_keystrokes		377.42	78.25	573.53
first_key_down_second_key_down_time	c, o	117.34	21.25	906.79
first_key_dwell_time	c	110.39	63.00	190.07
first_key_up_second_key_down_time	c, o	6.95	-122.59	832.35
second_key_dwell_time	c, o	130.44	31.00	248.87
digraph_duration	c, o	247.78	119.96	998.68
first_key_down_second_key_down_time	o, n	117.56	46.00	528.17
first_key_dwell_time	o	130.44	31.00	248.87
first_key_up_second_key_down_time	o, n	-12.88	-137.96	448.00
second_key_dwell_time	o, n	110.11	40.20	175.72
digraph_duration	o, n	227.67	152.03	632.15
first_key_down_second_key_down_time	n, s	128.49	40.20	880.01
first_key_dwell_time	n	110.11	40.20	175.72
first_key_up_second_key_down_time	n, s	18.38	-69.09	784.01
second_key_dwell_time	n, s	109.40	47.85	207.78
digraph_duration	n, s	237.89	112.10	960.00
first_key_down_second_key_down_time	s, o	143.65	43.08	1295.99
first_key_dwell_time	s	109.40	47.85	207.78
first_key_up_second_key_down_time	s, o	34.25	-149.29	1208.17
second_key_dwell_time	s, o	94.04	23.67	144.00
digraph_duration	s, o	237.69	125.00	1367.93
first_key_down_second_key_down_time	o, l	197.84	134.90	567.97
first_key_dwell_time	o	94.04	23.67	144.00
first_key_up_second_key_down_time	o, l	103.80	16.00	432.28
second_key_dwell_time	o, l	93.53	39.67	151.99
digraph_duration	o, l	291.37	191.68	639.70
first_key_down_second_key_down_time	l, e	143.00	74.48	639.90
first_key_dwell_time	l	93.53	39.67	151.99
first_key_up_second_key_down_time	l, e	49.47	-56.33	600.23
second_key_dwell_time	l, e	100.73	31.33	175.77
digraph_duration	l, e	243.73	143.90	711.67
first_key_down_second_key_down_time	e, .	193.32	55.97	1096.00
first_key_dwell_time	e	100.73	31.33	175.77

Table E.5: Mean, min and max values for aggregated frustrated feature vectors

Feature	Keys	Frustrated		
		Mean	Min	Max
first_key_up_second_key_down_time	e, .	92.59	-67.76	1008.03
second_key_dwell_time	e, .	100.15	31.79	181.19
digraph_duration	e, .	293.47	141.00	1168.00
first_key_down_second_key_down_time	., l	255.58	140.00	1199.95
first_key_dwell_time	.	100.15	31.79	181.19
first_key_up_second_key_down_time	., l	155.42	50.02	1120.00
second_key_dwell_time	., l	75.02	47.00	154.59
digraph_duration	., l	330.60	203.00	1272.03
first_key_down_second_key_down_time	l, o	194.36	124.80	1063.98
first_key_dwell_time	l	75.02	47.00	154.59
first_key_up_second_key_down_time	l, o	119.34	31.19	1000.02
second_key_dwell_time	l, o	96.75	55.82	175.79
digraph_duration	l, o	291.11	208.07	1143.80
first_key_down_second_key_down_time	o, g	105.69	39.96	208.14
first_key_dwell_time	o	96.75	55.82	175.79
first_key_up_second_key_down_time	o, g	8.94	-90.48	136.12
second_key_dwell_time	o, g	95.46	31.00	172.34
digraph_duration	o, g	201.15	112.07	334.77
first_key_down_second_key_down_time	g, Shift	395.47	80.07	7609.00
first_key_dwell_time	g	95.46	31.00	172.34
first_key_up_second_key_down_time	g, Shift	300.01	-15.80	7578.00
second_key_dwell_time	g, Shift	455.30	143.99	2320.07
digraph_duration	g, Shift	850.77	240.48	8359.00
first_key_down_second_key_down_time	Shift, (161.96	26.63	599.99
first_key_dwell_time	Shift	455.30	143.99	2320.07
first_key_up_second_key_down_time	Shift, (-293.34	-2224.00	-32.60
second_key_dwell_time	Shift, (108.75	48.00	191.80
digraph_duration	Shift, (270.71	110.00	712.00
first_key_down_second_key_down_time	c, o	117.34	21.25	906.79
first_key_dwell_time	c	110.39	63.00	190.07
first_key_up_second_key_down_time	c, o	6.95	-122.59	832.35
second_key_down_third_key_down_time	o, n	117.56	46.00	528.17
second_key_dwell_time	o	130.44	31.00	248.87
second_key_up_third_key_down_time	o, n	-12.88	-137.96	448.00
third_key_dwell_time	n	110.11	40.20	175.72
trigraph_duration	c, o, n	345.00	231.10	1166.22

Table E.5: Mean, min and max values for aggregated frustrated feature vectors

Feature	Keys	Frustrated		
		Mean	Min	Max
first_key_down_second_key_down_time	o, n	117.56	46.00	528.17
first_key_dwell_time	o	130.44	31.00	248.87
first_key_up_second_key_down_time	o, n	-12.88	-137.96	448.00
second_key_down_third_key_down_time	n, s	128.49	40.20	880.01
second_key_dwell_time	n	110.11	40.20	175.72
second_key_up_third_key_down_time	n, s	18.38	-69.09	784.01
third_key_dwell_time	s	109.40	47.85	207.78
trigraph_duration	o, n, s	355.45	205.48	1111.98
first_key_down_second_key_down_time	n, s	128.49	40.20	880.01
first_key_dwell_time	n	110.11	40.20	175.72
first_key_up_second_key_down_time	n, s	18.38	-69.09	784.01
second_key_down_third_key_down_time	s, o	143.65	43.08	1295.99
second_key_dwell_time	s	109.40	47.85	207.78
second_key_up_third_key_down_time	s, o	34.25	-149.29	1208.17
third_key_dwell_time	o	94.04	23.67	144.00
trigraph_duration	n, s, o	366.18	209.36	1439.88
first_key_down_second_key_down_time	s, o	143.65	43.08	1295.99
first_key_dwell_time	s	109.40	47.85	207.78
first_key_up_second_key_down_time	s, o	34.25	-149.29	1208.17
second_key_down_third_key_down_time	o, l	197.84	134.90	567.97
second_key_dwell_time	o	94.04	23.67	144.00
second_key_up_third_key_down_time	o, l	103.80	16.00	432.28
third_key_dwell_time	l	93.53	39.67	151.99
trigraph_duration	s, o, l	435.02	279.87	1576.11
first_key_down_second_key_down_time	o, l	197.84	134.90	567.97
first_key_dwell_time	o	94.04	23.67	144.00
first_key_up_second_key_down_time	o, l	103.80	16.00	432.28
second_key_down_third_key_down_time	l, e	143.00	74.48	639.90
second_key_dwell_time	l	93.53	39.67	151.99
second_key_up_third_key_down_time	l, e	49.47	-56.33	600.23
third_key_dwell_time	e	100.73	31.33	175.77
trigraph_duration	o, l, e	441.57	296.67	863.68
first_key_down_second_key_down_time	l, e	143.00	74.48	639.90
first_key_dwell_time	l	93.53	39.67	151.99
first_key_up_second_key_down_time	l, e	49.47	-56.33	600.23
second_key_down_third_key_down_time	e, .	193.32	55.97	1096.00

Table E.5: Mean, min and max values for aggregated frustrated feature vectors

Feature	Keys	Frustrated		
		Mean	Min	Max
second_key_dwell_time	e	100.73	31.33	175.77
second_key_up_third_key_down_time	e, .	92.59	-67.76	1008.03
third_key_dwell_time	.	100.15	31.79	181.19
trigraph_duration	l, e, .	436.47	234.00	1296.03
first_key_down_second_key_down_time	e, .	193.32	55.97	1096.00
first_key_dwell_time	e	100.73	31.33	175.77
first_key_up_second_key_down_time	e, .	92.59	-67.76	1008.03
second_key_down_third_key_down_time	., l	255.58	140.00	1199.95
second_key_dwell_time	.	100.15	31.79	181.19
second_key_up_third_key_down_time	., l	155.42	50.02	1120.00
third_key_dwell_time	l	75.02	47.00	154.59
trigraph_duration	e, ., l	523.91	288.02	1776.08
first_key_down_second_key_down_time	., l	255.58	140.00	1199.95
first_key_dwell_time	.	100.15	31.79	181.19
first_key_up_second_key_down_time	., l	155.42	50.02	1120.00
second_key_down_third_key_down_time	l, o	194.36	124.80	1063.98
second_key_dwell_time	l	75.02	47.00	154.59
second_key_up_third_key_down_time	l, o	119.34	31.19	1000.02
third_key_dwell_time	o	96.75	55.82	175.79
trigraph_duration	., l, o	546.68	359.00	1439.94
first_key_down_second_key_down_time	l, o	194.36	124.80	1063.98
first_key_dwell_time	l	75.02	47.00	154.59
first_key_up_second_key_down_time	l, o	119.34	31.19	1000.02
second_key_down_third_key_down_time	o, g	105.69	39.96	208.14
second_key_dwell_time	o	96.75	55.82	175.79
second_key_up_third_key_down_time	o, g	8.94	-90.48	136.12
third_key_dwell_time	g	95.46	31.00	172.34
trigraph_duration	l, o, g	395.51	296.00	1248.11
first_key_down_second_key_down_time	o, g	105.69	39.96	208.14
first_key_dwell_time	o	96.75	55.82	175.79
first_key_up_second_key_down_time	o, g	8.94	-90.48	136.12
second_key_down_third_key_down_time	g, Shift	395.47	80.07	7609.00
second_key_dwell_time	g	95.46	31.00	172.34
second_key_up_third_key_down_time	g, Shift	300.01	-15.80	7578.00
third_key_dwell_time	Shift	455.30	143.99	2320.07
trigraph_duration	o, g, Shift	956.46	344.59	8516.00

Table E.5: Mean, min and max values for aggregated frustrated feature vectors

Feature	Keys	Frustrated		
		Mean	Min	Max
first_key_down_second_key_down_time	g, Shift	395.47	80.07	7609.00
first_key_dwell_time	g	95.46	31.00	172.34
first_key_up_second_key_down_time	g, Shift	300.01	-15.80	7578.00
second_key_down_third_key_down_time	Shift, (161.96	26.63	599.99
second_key_dwell_time	Shift	455.30	143.99	2320.07
second_key_up_third_key_down_time	Shift, (-293.34	-2224.00	-32.60
third_key_dwell_time	(108.75	48.00	191.80
trigraph_duration	g, Shift, (666.18	203.00	7968.00

Table E.6: Mean, min and max values for aggregated surprised feature vectors

Feature	Keys	Surprised		
		Mean	Min	Max
number_of_misses		6.57	0.00	15.00
avg_dwell		129.67	75.69	211.27
avg_flight		91.91	0.84	411.36
avg_keystrokes		343.51	110.26	520.00
first_key_down_second_key_down_time	c, o	86.36	26.72	225.23
first_key_dwell_time	c	116.89	78.00	202.30
first_key_up_second_key_down_time	c, o	-30.53	-121.59	113.36
second_key_dwell_time	c, o	106.15	62.00	174.88
digraph_duration	c, o	192.51	120.02	342.84
first_key_down_second_key_down_time	o, n	620.05	96.02	3438.89
first_key_dwell_time	o	106.15	62.00	174.88
first_key_up_second_key_down_time	o, n	513.90	-74.65	3321.28
second_key_dwell_time	o, n	90.90	29.67	159.75
digraph_duration	o, n	710.95	176.02	3547.40
first_key_down_second_key_down_time	n, s	105.32	48.02	221.81
first_key_dwell_time	n	90.90	29.67	159.75
first_key_up_second_key_down_time	n, s	14.43	-63.86	73.91
second_key_dwell_time	n, s	121.22	61.40	191.55
digraph_duration	n, s	226.54	143.99	413.36
first_key_down_second_key_down_time	s, o	294.79	64.43	1387.11
first_key_dwell_time	s	121.22	61.40	191.55
first_key_up_second_key_down_time	s, o	173.58	-85.33	1248.29
second_key_dwell_time	s, o	83.99	56.06	122.48

Table E.6: Mean, min and max values for aggregated surprised feature vectors

Feature	Keys	Surprised		
		Mean	Min	Max
digraph_duration	s, o	378.78	122.79	1499.12
first_key_down_second_key_down_time	o, l	164.43	80.02	242.05
first_key_dwell_time	o	83.99	56.06	122.48
first_key_up_second_key_down_time	o, l	80.44	23.96	125.90
second_key_dwell_time	o, l	107.94	62.00	133.65
digraph_duration	o, l	272.37	199.98	364.54
first_key_down_second_key_down_time	l, e	186.10	94.00	557.15
first_key_dwell_time	l	107.94	62.00	133.65
first_key_up_second_key_down_time	l, e	78.15	-31.82	434.67
second_key_dwell_time	l, e	104.28	62.00	175.54
digraph_duration	l, e	290.38	156.00	732.70
first_key_down_second_key_down_time	e, .	162.99	93.00	292.26
first_key_dwell_time	e	104.28	62.00	175.54
first_key_up_second_key_down_time	e, .	58.71	5.34	159.05
second_key_dwell_time	e, .	92.86	47.00	138.42
digraph_duration	e, .	255.84	140.00	388.53
first_key_down_second_key_down_time	., l	205.36	185.78	253.02
first_key_dwell_time	.	92.86	47.00	138.42
first_key_up_second_key_down_time	., l	112.51	66.18	162.44
second_key_dwell_time	., l	74.68	40.01	95.94
digraph_duration	., l	280.04	247.91	332.81
first_key_down_second_key_down_time	l, o	170.81	150.95	203.00
first_key_dwell_time	l	74.68	40.01	95.94
first_key_up_second_key_down_time	l, o	96.13	55.02	141.00
second_key_dwell_time	l, o	100.83	48.09	154.57
digraph_duration	l, o	271.64	208.01	310.29
first_key_down_second_key_down_time	o, g	148.85	74.69	224.08
first_key_dwell_time	o	100.83	48.09	154.57
first_key_up_second_key_down_time	o, g	48.01	-79.89	175.98
second_key_dwell_time	o, g	101.56	71.13	165.12
digraph_duration	o, g	250.40	188.00	320.12
first_key_down_second_key_down_time	g, Shift	420.35	125.00	632.02
first_key_dwell_time	g	101.56	71.13	165.12
first_key_up_second_key_down_time	g, Shift	318.79	46.00	560.89
second_key_dwell_time	g, Shift	480.71	157.00	1017.15
digraph_duration	g, Shift	901.06	282.00	1539.21

Table E.6: Mean, min and max values for aggregated surprised feature vectors

Feature	Keys	Surprised		
		Mean	Min	Max
first_key_down_second_key_down_time	Shift, (211.47	23.91	555.26
first_key_dwell_time	Shift	480.71	157.00	1017.15
first_key_up_second_key_down_time	Shift, (-269.25	-692.42	-110.00
second_key_dwell_time	Shift, (103.66	55.58	154.42
digraph_duration	Shift, (315.12	110.00	610.83
first_key_down_second_key_down_time	c, o	86.36	26.72	225.23
first_key_dwell_time	c	116.89	78.00	202.30
first_key_up_second_key_down_time	c, o	-30.53	-121.59	113.36
second_key_down_third_key_down_time	o, n	620.05	96.02	3438.89
second_key_dwell_time	o	106.15	62.00	174.88
second_key_up_third_key_down_time	o, n	513.90	-74.65	3321.28
third_key_dwell_time	n	90.90	29.67	159.75
trigraph_duration	c, o, n	797.31	224.01	3772.63
first_key_down_second_key_down_time	o, n	620.05	96.02	3438.89
first_key_dwell_time	o	106.15	62.00	174.88
first_key_up_second_key_down_time	o, n	513.90	-74.65	3321.28
second_key_down_third_key_down_time	n, s	105.32	48.02	221.81
second_key_dwell_time	n	90.90	29.67	159.75
second_key_up_third_key_down_time	n, s	14.43	-63.86	73.91
third_key_dwell_time	s	121.22	61.40	191.55
trigraph_duration	o, n, s	846.59	240.00	3679.42
first_key_down_second_key_down_time	n, s	105.32	48.02	221.81
first_key_dwell_time	n	90.90	29.67	159.75
first_key_up_second_key_down_time	n, s	14.43	-63.86	73.91
second_key_down_third_key_down_time	s, o	294.79	64.43	1387.11
second_key_dwell_time	s	121.22	61.40	191.55
second_key_up_third_key_down_time	s, o	173.58	-85.33	1248.29
third_key_dwell_time	o	83.99	56.06	122.48
trigraph_duration	n, s, o	484.10	240.08	1600.82
first_key_down_second_key_down_time	s, o	294.79	64.43	1387.11
first_key_dwell_time	s	121.22	61.40	191.55
first_key_up_second_key_down_time	s, o	173.58	-85.33	1248.29
second_key_down_third_key_down_time	o, l	164.43	80.02	242.05
second_key_dwell_time	o	83.99	56.06	122.48
second_key_up_third_key_down_time	o, l	80.44	23.96	125.90
third_key_dwell_time	l	107.94	62.00	133.65

Table E.6: Mean, min and max values for aggregated surprised feature vectors

Feature	Keys	Surprised		
		Mean	Min	Max
trigraph_duration	s, o, l	567.16	320.01	1656.85
first_key_down_second_key_down_time	o, l	164.43	80.02	242.05
first_key_dwell_time	o	83.99	56.06	122.48
first_key_up_second_key_down_time	o, l	80.44	23.96	125.90
second_key_down_third_key_down_time	l, e	186.10	94.00	557.15
second_key_dwell_time	l	107.94	62.00	133.65
second_key_up_third_key_down_time	l, e	78.15	-31.82	434.67
third_key_dwell_time	e	104.28	62.00	175.54
trigraph_duration	o, l, e	454.80	272.01	974.75
first_key_down_second_key_down_time	l, e	186.10	94.00	557.15
first_key_dwell_time	l	107.94	62.00	133.65
first_key_up_second_key_down_time	l, e	78.15	-31.82	434.67
second_key_down_third_key_down_time	e, .	162.99	93.00	292.26
second_key_dwell_time	e	104.28	62.00	175.54
second_key_up_third_key_down_time	e, .	58.71	5.34	159.05
third_key_dwell_time	.	92.86	47.00	138.42
trigraph_duration	l, e, .	441.94	234.00	918.15
first_key_down_second_key_down_time	e, .	162.99	93.00	292.26
first_key_dwell_time	e	104.28	62.00	175.54
first_key_up_second_key_down_time	e, .	58.71	5.34	159.05
second_key_down_third_key_down_time	., l	205.36	185.78	253.02
second_key_dwell_time	.	92.86	47.00	138.42
second_key_up_third_key_down_time	., l	112.51	66.18	162.44
third_key_dwell_time	l	74.68	40.01	95.94
trigraph_duration	e, ., l	443.02	343.00	603.23
first_key_down_second_key_down_time	., l	205.36	185.78	253.02
first_key_dwell_time	.	92.86	47.00	138.42
first_key_up_second_key_down_time	., l	112.51	66.18	162.44
second_key_down_third_key_down_time	l, o	170.81	150.95	203.00
second_key_dwell_time	l	74.68	40.01	95.94
second_key_up_third_key_down_time	l, o	96.13	55.02	141.00
third_key_dwell_time	o	100.83	48.09	154.57
trigraph_duration	., l, o	477.00	415.91	538.07
first_key_down_second_key_down_time	l, o	170.81	150.95	203.00
first_key_dwell_time	l	74.68	40.01	95.94
first_key_up_second_key_down_time	l, o	96.13	55.02	141.00

Table E.6: Mean, min and max values for aggregated surprised feature vectors

Feature	Keys	Surprised		
		<i>Mean</i>	<i>Min</i>	<i>Max</i>
second_key_down_third_key_down_time	o, g	148.85	74.69	224.08
second_key_dwell_time	o	100.83	48.09	154.57
second_key_up_third_key_down_time	o, g	48.01	-79.89	175.98
third_key_dwell_time	g	101.56	71.13	165.12
trigraph_duration	l, o, g	421.21	348.19	514.66
first_key_down_second_key_down_time	o, g	148.85	74.69	224.08
first_key_dwell_time	o	100.83	48.09	154.57
first_key_up_second_key_down_time	o, g	48.01	-79.89	175.98
second_key_down_third_key_down_time	g, Shift	420.35	125.00	632.02
second_key_dwell_time	g	101.56	71.13	165.12
second_key_up_third_key_down_time	g, Shift	318.79	46.00	560.89
third_key_dwell_time	Shift	480.71	157.00	1017.15
trigraph_duration	o, g, Shift	1049.91	391.00	1673.63
first_key_down_second_key_down_time	g, Shift	420.35	125.00	632.02
first_key_dwell_time	g	101.56	71.13	165.12
first_key_up_second_key_down_time	g, Shift	318.79	46.00	560.89
second_key_down_third_key_down_time	Shift, (211.47	23.91	555.26
second_key_dwell_time	Shift	480.71	157.00	1017.15
second_key_up_third_key_down_time	Shift, (-269.25	-692.42	-110.00
third_key_dwell_time	(103.66	55.58	154.42
trigraph_duration	g, Shift, (735.47	235.00	1242.85

Bibliography

- Bahreini, K., Nadolski, R., and Westera, W. (2016a). Data fusion for real-time multimodal emotion recognition through webcams and microphones in e-learning. *International Journal of Human-Computer Interaction*, 32(5):415–430.
- Bahreini, K., Nadolski, R., and Westera, W. (2016b). Towards multimodal emotion recognition in e-learning environments. *Interactive Learning Environments*, 24(3):590–605.
- Bahreini, K., Nadolski, R., and Westera, W. (2016c). Towards real-time speech emotion recognition for affective e-learning. *Education and Information Technologies*, 21(5):1367–1386.
- Baker, R. S., D’Mello, S. K., Rodrigo, M. M. T., and Graesser, A. C. (2010). Better to be frustrated than bored: The incidence, persistence, and impact of learners’ cognitive–affective states during interactions with three different computer-based learning environments. *International Journal of Human-Computer Studies*, 68(4):223–241.
- Beckmann, M., Ebecken, N. F., and de Lima, B. S. P. (2015). A knn undersampling approach for data balancing. *Journal of Intelligent Learning Systems and Applications*, 7(4):104.
- Bjørndal, B. and Bakken, J. B. (2015). Overvåker deg når du taster (dn+) - dn.no. <http://www.dn.no/nyheter/politikkSamfunn/2015/08/21/2142/IT/overvker-deg-nr-du-taster>. Accessed: 2017-05-25.
- Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122.

- Cannon, W. B. (1927). The james-lange theory of emotions: A critical examination and an alternative theory. *The American journal of psychology*, 39(1/4):106–124.
- Carroll, L. (1898). *Alice’s Adventures in Wonderland*. Macmillan’s sixpenny series. Macmillan.
- Choppin, A. (2000). *EEG-based human interface for disabled individuals: Emotion expression with neural networks*. PhD thesis, Tokyo institute of technology.
- Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.
- Csikszentmihalyi, M. (1990). Flow: The psychology of optimal performance. NY: Cambridge University Press.
- Dunstone, T. and Yager, N. (2008). *Biometric system and data analysis: Design, evaluation, and data mining*. Springer Science & Business Media.
- Ekman, P., Friesen, W. V., and Hager, J. (1978). The facial action coding system (facs): A technique for the measurement of facial action. palo alto. CA: Consulting Psychologists Press, Inc. Ekman, P. Levenson. RW, & Friesen WV (1983). Auto-nomic nervous system activity distinguishes among emotions. *Science*, 221:1208–12.
- Epp, C., Lippold, M., and Mandryk, R. L. (2011). Identifying emotional states using keystroke dynamics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 715–724. ACM.
- Epp, C. C. (2010). *Identifying emotional states through keystroke dynamics*. PhD thesis, Saskatchewan.
- Fitbit (2016). Fitbit charge 2™ heart rate + fitness wristband. <https://www.fitbit.com/charge2>. Accessed: 2017-05-13.
- Gaines, R. S., Lisowski, W., Press, S. J., and Shapiro, N. (1980). Authentication by keystroke timing: Some preliminary results. Technical report, DTIC Document.
- Graesser, A., D’MELLO, S., Chipman, P., King, B., and McDANIEL, B. (2007). Exploring relationships between affect and learning with autotutor. In *Proc Int Conf AIED*.
- Graesser, A. C., Conley, M. W., and Olney, A. (2012). *Intelligent tutoring systems*. American Psychological Association.

- Hall, M. A. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato.
- Haverbeke, M. (2014). *Eloquent javascript: A modern introduction to programming*. No Starch Press.
- Hearn, T. (2013). thearn/webcam-pulse-detector: A python application that detects and highlights the heart-rate of an individual (using only their own webcam) in real-time. <https://github.com/thearn/webcam-pulse-detector>. Accessed: 2017-03-08.
- Hernandez, J., Paredes, P., Roseway, A., and Czerwinski, M. (2014). Under pressure: sensing stress of computer users. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 51–60. ACM.
- Kao, F. C., Wang, S. P. R., Huang, C. H., Chen, C. C., and Lin, Y. K. (2015). Brainwave analysis of positive and negative emotions. In *Management, Information and Educational Engineering: Proceedings of the 2014 International Conference on Management, Information and Educational Engineering (MIEE 2014), Xiamen, China, November 22-23, 2014*, pages 1263–1266. CRC Press.
- KM, A. K., Kiran, B., Shreyas, B., and Victor, S. J. (2015). A multimodal approach to detect user’s emotion. *Procedia Computer Science*, 70:296–303.
- Kołakowska, A. (2013). A review of emotion recognition methods based on keystroke dynamics and mouse movements. In *Human System Interaction (HSI), 2013 The 6th International Conference on*, pages 548–555. IEEE.
- Kołakowska, A. (2015). Recognizing emotions on the basis of keystroke dynamics. In *Human System Interactions (HSI), 2015 8th International Conference on*, pages 291–297. IEEE.
- Kołakowska, A. (2016). Towards detecting programmers’ stress on the basis of keystroke dynamics. In *Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on*, pages 1621–1626. IEEE.
- Kort, B., Reilly, R., and Picard, R. W. (2001). An affective model of interplay between emotions and learning: Reengineering educational pedagogy-building a learning companion. In *Advanced Learning Technologies, 2001. Proceedings. IEEE International Conference on*, pages 43–46. IEEE.
- Lang, P. J. (1995). The emotion probe: Studies of motivation and attention. *American psychologist*, 50(5):372.

- Leinonen, J., Longi, K., Klami, A., and Vihavainen, A. (2016). Automatic inference of programming performance and experience from typing patterns. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 132–137. ACM.
- Longi, K., Leinonen, J., Nygren, H., Salmi, J., Klami, A., and Vihavainen, A. (2015). Identification of programmers from typing patterns. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, pages 60–67. ACM.
- Ma, W., Adesope, O. O., Nesbit, J. C., and Liu, Q. (2014). Intelligent tutoring systems and learning outcomes: A meta-analysis.
- Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Education - Europe.
- Mondal, S. and Bours, P. (2014). Continuous authentication using fuzzy logic. In *Proceedings of the 7th international conference on security of information and networks*, page 231. ACM.
- Monrose, F. and Rubin, A. (1997). Authentication via keystroke dynamics. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 48–56. ACM.
- Monrose, F. and Rubin, A. D. (2000). Keystroke dynamics as a biometric for authentication. *Future Generation computer systems*, 16(4):351–359.
- Oates, B. J. (2005). *Researching information systems and computing*. Sage.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- Perkins, R. E. and Hill, A. (1985). Cognitive and affective aspects of boredom. *British Journal of Psychology*, 76(2):221–234.
- Picard, R. W. (1997). *Affective computing*, volume 252. MIT press Cambridge.
- Revett, K., Gorunescu, F., Gorunescu, M., Ene, M., Magalhaes, S., and Santos, H. (2007). A machine learning approach to keystroke dynamics based user authentication. *International Journal of Electronic Security and Digital Forensics*, 1(1):55–70.
- Russell, J. A. (2003). Core affect and the psychological construction of emotion. *Psychological review*, 110(1):145.

- Salazar, V. E., Lucio, N. D., and Funk, M. D. (2017). Accuracy of fitbit charge 2 worn at different wrist locations during exercise. In *International Journal of Exercise Science: Conference Proceedings*, volume 2, page 41.
- Sarrafzadeh, A., Alexander, S., Dadgostar, F., Fan, C., and Bigdeli, A. (2008). “how do you know that i don’t understand?” a look at the future of intelligent tutoring systems. *Computers in Human Behavior*, 24(4):1342–1363.
- Shukla, P. and Solanki, R. (2013). Web based keystroke dynamics application for identifying emotional state. *Institute of Engineering and Technology, DAVV, Indore, India in Nov.*
- Solanki, R. and Shukla, P. (2014). Estimation of the user’s emotional state by keystroke dynamics. *International Journal of Computer Applications*, 94(13).
- Sung, K.-s. and Cho, S. (2006). Ga svm wrapper ensemble for keystroke dynamics authentication. *International Conference on Biometrics, Hong Kong*, pages 654–660.
- Teh, P. S., Teoh, A. B. J., and Yue, S. (2013). A survey of keystroke dynamics biometrics. *The Scientific World Journal*, 2013.
- Villani, M., Tappert, C., Ngo, G., Simone, J., Fort, H. S., and Cha, S.-H. (2006). Keystroke biometric recognition studies on long-text input under ideal and application-oriented conditions. In *Computer Vision and Pattern Recognition Workshop, 2006. CVPRW’06. Conference on*, pages 39–39. IEEE.
- Witten, I. H., Frank, E., Hall, M. A., and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.
- Young, T. A. (2012). s3cur3/keystrokedynamics.js: A proof-of-concept for web authentication via keystroke dynamics (that is, authenticating users based on *how* they type rather than *what* they type). <https://github.com/s3cur3/KeystrokeDynamics.JS>. Accessed: 2017-01-19.
- Yuksel, B. F., Oleson, K. B., Harrison, L., Peck, E. M., Afergan, D., Chang, R., and Jacob, R. J. (2016). Learn piano with bach: An adaptive learning interface that adjusts task difficulty based on brain state. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 5372–5384. ACM.
- Zimmermann, P., Gomez, P., Danuser, B., and Schär, S. (2006). Extending usability: putting affect into the user-experience. *Proceedings of NordiCHI’06*, pages 27–32.