NTNU

Norwegian University of
Science and Technology

A Shallow Neural Network Architecture for Native Language Identification

Hans Olav Slotte

Master of Science in Computer Science
Submission date: March 2018
Supervisor: Björn Gambäck, IDI

Norwegian University of Science and Technology
Department of Computer Science

Hans Olav Slotte
slotte@stud.ntnu.no

A Shallow Neural Network Architecture for Native Language Identification

Master Thesis, March 2018

Supervised by Björn Gambäck

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

Problem Description

The problem description as it was given by Björn Gambäck was stated as:

> Native Language Identification is the task of identifying the native language of a writer based solely on a sample of their writing in another language. The task is typically framed as a classification problem where the set of native languages is known beforehand. Most work has focused on identifying the native language of writers learning English as a second language. The master thesis work connects to previous work in IDI's AI group and potentially involves participation in a "shared task competition" on Native Language Identification where training and test data is made available by the organisers (such as https://sites.google.com/site/nlisharedtask2013/).

Abstract

Native Language Identification (NLI) is the task of identifying writers' or speakers' native languages faced only with text or speech in another language. This task has been part of research into Second Language Acquisition for a long time, and has more recently also become a part of Natural Language Processing research. This research has resulted in two shared tasks where teams from around the world have compared results and methods.

So far, it has mostly been traditional statistical machine learing methods that have been used in these tasks, although there have been a few attempts at using neural methods. This thesis looks at using Multilayer Perceptron (MLP) classifiers for NLI both on their own and as part of an ensemble classifier as a possible approach to this task. During experimentation, these are compared to a more traditional classifier type used in NLI, Support Vector Machines (SVM). The experiments found consistently improved results when using a two layer MLP classifier compared to a linear SVM classifier, and gave mid range results compared to the systems submitted to the 2017 Shared Task, while using fairly small feature vectors.

One of the problems that was brought up at the 2017 Shared Task was the fact that classification systems could have trouble generalizing to texts written about subjects on which the classifier had not been trained. Therefore, this thesis also looks at whether a simpler preprocessing method can give better results for such unseen topics than the more traditional input preprocessing methods. Two different methods are initially compared, and experiments showed that for a word n-gram feature vector, the simpler method had better results, while the opposite is true for character and part-of-speech feature vectors. This leads to a mixed preprocessing method that had better results than the traditional method for most tested prompts.

Sammendrag

Morsmålsidentifisering går ut på å identifisere en forfatters eller talers morsmål basert på tekster eller opptak gjort av denne personen på et annet språk. Dette har vært et forskningsområde innenfor språklæring lenge, siden det å identifisere språkfeil som er mer vanlige for et gitt morsmål kan hjelpe til å forme et mer rettet læringsopplegg. Mer nylig har morsmålsidentifisering også blitt et forskningsfelt innen naturlig språkbehandling. Denne forskningen har kulminert i to fellesoppgaver, der grupper fra rundt om i verden har sammenlignet resultater og metoder.

Hittil har det i all hovedsak vært tradisjonelle statistiske maskinlæringsmetoder som har blitt brukt i disse oppgavene, selv om det har vært mindre forsøk på å bruke nevrale metoder. Denne avhandlingen ser på flerlagsperceptroner som en mulig alternativ metode, både som enkeltklassifikator og som en del av et større klassifikatorensemble. I eksperimenter ble disse sammenlignet med den mer tradisjonelle støttevektormaskin-metoden, hvor en tolags perceptronklassifikator fikk konsekvent bedre resultater enn en lineær støttevektormaskin. Resultatene er også sammenlignbare med systemer i tredje gruppe i fellesoppgaven i 2017.

I den siste fellesoppgaven i 2017, ble det brakt opp at klassifikatorer kunne ha problemer med å generalisere til tekster som hadde blitt skrevet om tema klassifikatoren ikke hadde trent på. Avhandlingen undersøker derfor også hvorvidt en enklere forbehandling av data kan forbedre en klassifikators resultater på slike nye tema. Dette starter med eksperimenter med to ulike forbehandlingsmetoder, og disse eksperimentene viste at den enkleste metoden gav bedre resultater for ett av de undersøkte trekkene (ord), mens den mer tradisjonelle metoden gav bedre resultater for de to andre (bokstav og ordklasse). Med disse resultatene foreslår avhandlingen en samlet forbehandlingsmetode som gav bedre resultater enn den tradisjonelle metoden for de fleste nye tema som ble testet.

Preface

This thesis is submitted as the final part of the work to achieve the degree of Master of Science in Computer Science from the Norwegian University of Science and Technology (NTNU). The work was done at the Department of Computer Science and the thesis work was supervised by Björn Gambäck.

Trondheim, 15 March 2018

Hans Olav Slotte

Acknowledgements

I would like to thank my parents, Mette Nilsen and Per Arne Slotte, for support through all my life and for always believing in me. I would also like to thank my supervisor, Björn Gambäck, for his input and advice through all the work on my thesis, and for the opportunity to work on this very interesting topic.

Contents

List of Figures

List of Tables

1 Introduction

When people learn a new language (L2), what impact does their native language (L1) have on the way they use the new language? This has been a question in Second Language Acquisition (SLA) for a long time. Lado (1957) introduced the concept of *contrastive linguistics*, which seeks to map the differences and similarities between pairs of languages. This method, also called *contrastive analysis*, has been used extensively to try to explain why some language features were more difficult for some learners to acquire than others. While it has been found that not all mistakes made when learning a new language can be explained by interference by a learner's L1, it is clear that some mistakes are more common among speakers of the same L1 than others.

With the increasingly global modern world, more and more people need to learn a second language to interact and work with people in other countries. Finding automated ways of identifying the mistakes common to the same L1 could help language learners and their teachers personalize their learning by keeping these mistakes in mind.

1.1 Background and Motivation

Native Language Identification was initially explored as an area of research for computer science as a continuation of research on another text classification task: Authorship Attribution. This task has many similarities to NLI, since both look for peculiarities in the way text is written in order to help uncover information about the author. Research was slow at first, and it did not really take off until the end of the previous decade. This decade, however, has seen much more active research, and in the last five years, there have been two shared tasks, most recently in 2017, and the accuracy of systems has steadily climbed. Many different approaches have been used, but almost all of them have been traditional statistical methods. Neural methods were only recently attempted, and the traditional methods have had the best results so far.

As mentioned, an NLI shared task was held at The Twelfth Workshop on Innovative Use of NLP for Building Educational Applications (BEA) in 2017. One problem that was brought up (Kulmizev et al. 2017) was that when systems are tested on a set that contain texts written for topic prompts for which no texts in the training set had been written, accuracy dropped (between 3% and 20% depending on the prompt for Kulmizev et al. (2017)). To get better systems, it is therefore important to find methods that help classifiers perform better on texts written about previously unseen topics.

1.2 Goal and Research Questions

The goal of this thesis is twofold. First, to explore approaches to Native Language Identification and attempt to find a simple classifier model that achieves good results. Second, to find an input preprocessing method that improves accuracy on texts written for prompts on which the classifier has not been trained compared to traditional input preprocessing methods.

1.2.1 New classifier model

Neural methods have been very successful at many different classification tasks, especially within computer vision and speech recognition. However, they have remained fairly unused in Native Language Identification, possibly due to the small amount of prepared data that exists compared to many other text classification tasks. This leads to the conclusion that an advanced neural model might very readily overfit the data, but Li and Zou (2017) note that multilayer perceptron (MLP) classifiers do get better results than linear support vector machines (SVM) in most cases for their system setup, at the cost of training speed. A simple neural model is able to keep the strength of separating data that is not linearly separable while still being simple enough that it is not hampered by overfitting. Therefore, it would be interesting to look at the effectiveness of simple feedforward neural networks for this task in combination with fairly small feature vectors to lessen the problem of the training speed. This thesis looks at one or more network models to experiment with, and finally propose an MLP design for the task. An important subgoal is finding out how many layers are ideal for the neural model.

1.2.2 Input preprocessing

While traditional methods of input preprocessing carefully picks out the most important terms for identification, inherent bias in the way different learners write about any given topics could lead to a large amount of terms that only give information for a text written about particular topics that are included in the training set. This might be one of the reasons that accuracy drops for traditional systems when faced with unseen topics. A simpler method would obviously be worse at classifying texts written about topics included in the training set, but not being as selective in the choice of terms might improve accuracy on texts written about unseen topics.

1.3 Research Questions

Q1: *To what extent can simple neural models be used effectively for NLI?*

Q2: *How many layers are enough for a trained MLP classifier to completely fit the training data?*

Q3: *To what extent can simpler preprocessing of input help classifiers perform better on texts written about topics which were not included in the training set?*

1.4 Contributions

C1: *A simple MLP/SVM ensemble classifier model for Native Language Identification. This model achieves a 10-fold cross-validation accuracy of* 80.41% *on essay-only input and* 87.25% *when also using speech data.*

C2: *An even simpler two-layer fully connected perceptron classifier for NLI. This model achieves a 10-fold cross-validation accuracy of* 82.56% *on essay-only input and* 75.69% *with speech data.*

C3: *A comparison of two different input preprocessing methods and the difference in accuracy when faced with texts written for previously unseen prompts. This ends with a mixed preprocessing method that has a lower drop in accuracy than the other two methods for most of the tested prompts.*

1.5 Thesis Structure

- **Section 2** goes through the background theory necessary to understand the rest of the thesis, as well as a including a list of the tools used in the system created as part of this thesis.
- **Section 3** gives a run through of the history of Native Language Identification (NLI) and the approaches that have been previously used, including a description of the current de facto standard dataset for training NLI classification models: TOEFL11. It ends with a description of the state of the art systems for this classification task.
- **Section 4** describes a proposed shallow neural network architecture for NLI.
- **Section 5** goes through the experimental setup, and shows the experimental results.
- **Section 6** contains a detailed discussion about the final results of these experiments.
- **Section 7** describes how this thesis fulfilled its objectives, gives some final thoughts on the experimental results, and gives a few suggestions for more explorations in this field.

2 Background Theory

It is important for a reader of this thesis to understand some basic concepts before continuing on with previous work in Native Language Identification, and even more importantly before moving on to the classifier proposed in section 4. Some concepts require only a cursory understanding, for example classifier types that have been used in systems discussed in section 3. Concepts that are directly related to the system proposed in this thesis are described in more detail.

2.1 Supervised Learning

NLI, as a subset of text classification, is a typical example of a field in which Supervised Learning methods are the logical choice for learning how to classify correctly. Supervised learning algorithms gain their knowledge of the domain space through examples provided by some teacher. Such algorithms work extremely well if you know the classes a domain space logically separates into, but you might not know exactly how to place an unseen example into one of these classes. The supervised learning algorithm is given the examples that are available, and attempts to make its guesses match the examples as closely as possible, while avoiding overfitting (see 2.4). When a model is created, it can be evaluated in relation to the observed data points by way of a *loss function*. This loss function is a single number (the loss) which shows how bad the model is, with a higher loss representing a worse model. This could be as simple as just counting the amount of wrong guesses, called the *0-1 loss function*, or more advanced, like the commonly used *cross entropy loss function*, which is described in detail in 2.3.

Common examples of supervised algorithms include Support Vector Machines (see 2.7.1) and Maximum Entropy Learning (also known as Logistic Regression). Different types of neural network designs are also commonly used for this kind of learning.

2.2 Classification Accuracy

When creating classifiers, it is important to find a good way of evaluating how well the classifier correctly places items into their correct class. The obvious way is to simply count how many items are correctly identified and divide that number by the total amount of items that were tested. This very simple method is called the *accuracy* of the classifier, and gives a fairly good idea of performance. However, it gives us very little information beyond that, for example in what way it is having trouble, which items it is having trouble with, etc. This leads us to two other measures, *precision* and *recall*.

Precision is the fraction of items predicted to have a certain classification that actually did have that classification. That is, the amount of true positives for a class divided by the total amount of positive predictions for that class. As such, it measures the extent to which a given class was given to items in the correct class as opposed to items in other classes. On the other hand, recall is the fraction of items that were correctly predicted for a certain label divided by the total items that actually belong to that label. That is, it measures to what extent a classifier correctly places all items in a given class into

the correct class. Precision and recall scores are calculated for each class separately, and gives you a lot of information regarding the problems faced for each class. For example, a class with high precision, but low recall is rarely predicted but when predicted, it is usually correct, while a class with low precision, but high recall is often predicted, but the prediction is often wrong.

For each class, these two numbers can be combined into a single measure called the *F-measure* or *F1*, which is the harmonic mean of these two numbers, given by

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

The harmonic mean is close to the arithmetic mean when the two numbers are close, but, since these numbers are always smaller than 1, punishes low scores more harshly than arithmetic mean, and as such gives a better idea of when a model is poor at either precision or recall than the accuracy score does.

As we can see, precision, recall, and F-measure are calculated on a per-class basis. How can we combine these scores into a single measure for all classes? There are two main ways to do this: *Micro-averaging* and *macro-averaging*. Micro-averaging counts the true positives, false positives, and false negatives for all classes, and calculates a new precision, recall, and F-measure for the new total. On the other hand, macro-averaging calculates the arithmetic mean of the precision and recall of all classes, and a new F-measure is calculated as normal using these two macro-averages scores. The averages given for the final experiment in this thesis are calculated using prevalence-weighted macro-averaging. Prevalence weighting means that the scores for each class are weighted by the amount of items in that class. This means that, unlike pure macro-averaging, label imbalance is taken into account.

2.3 Cross Entropy Loss Function

The multilayer perceptron classifier proposed in this thesis uses the cross entropy loss function to evaluate the model. To understand what cross entropy is, it is first necessary to understand what information entropy is. Say you are trying to encode an observation using the least possible amount of bits on average. Such an optimal encoding would mean that each bit in the observation gives the most amount of information possible. Obviously, you would ideally want to use fewer bits encoding an observation that is common, and more bits for rare observations. Given that we know how to encode the information, and this model is given by the probability distribution y, we can find the entropy of the system as

$$E(y) = \sum_i y_i \log \frac{1}{y_i} = -\sum_i y_i \log y_i \tag{2}$$

where y_i the probability of a given observation. Now, let us say that we do not know how to map our observations to a probability distribution for encoding. Given the two

probability distributions y and $\hat{y}$, the cross entropy between these two distributions is given by

$$CE(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i \tag{3}$$

where y can be thought of as the training example, i.e. the true probability distribution for this observation, while $\hat{y}$ is the probability distribution produced by the machine learning model that is being trained. As an example, given the true label vector $y = [0, 0, 1, 0]$ and $\hat{y} = [0.2, 0.5, 0.3, 0.1]$, the cross entropy of the observation is given as

$$CE = -\sum_i y_i \log \hat{y}_i = -1 \cdot \log 0.3 \approx 1.20397 \tag{4}$$

As we can see, given that a label is given as a probability vector where only one item will have a value, cross entropy only looks at the probability the model being trained gives for that one item. Just like a model being trained using *1-0 loss* tries to minimize the amount of differences between the model and the true labels, a model being trained using *cross entropy loss* tries to minimize this cross entropy, since a model that gives the same probability distribution as the true labels, will have a cross entropy of 0. Cross entropy loss is currently the most common loss function in neural network classifiers.

2.4 Overfitting and Regularization

When a learning algorithm tries to learn from a set of examples, it can be prone to matching the training data too closesly. This is known as overfitting, and results in poor generalization to unseen data. In order to avoid this problem, it is necessary to somehow penalize complicated models over simple ones, or to make the amount of training data so large and representative that overfitting is less of a problem. For the first, the traditional approach is something known as *regularization*. Regularization adds an extra term to the loss function which penalizes the weights of the model, either as the sum of the weights (L1 regularization) or as the sum of the squared weights (L2 regularization). For neural networks, another viable method of avoiding overfitting is to randomly choose a subset of the weights at each training step and ignore them for that step, so that each variable in the model is less dependent on other variables to give valuable information. This method is known as *dropout* (Srivastava et al. 2014).

2.5 Validating Machine Learning Models

In order to know the efficacy of a Machine Learning system, it is necessary to somehow validate the resulting model in a way that tells us its ability to generalize to new data points. The simplest way of doing this is to split a dataset into two parts, one larger one to use as a training set and a smaller one to use as a test set once training has completed. This is called *hold-out validation*, and is a very common measure, especially during early experimentation, precisely because it is simple to implement, and you get a single accuracy quickly.

The second main option, which will give a more accurate evaluation of the design at the expense of time efficiency, is *cross-validation*. Cross validation is essentially repeating hold-out validation several times and averaging the results. As an example, when performing 10-fold cross-validation, the data set is split into 10 parts, and 10 models are trained using each of the 10 parts as the test set once and the rest as the training set for that model. Then the test results for each of these 10 models is averaged and used as the results for the machine learning design.

The most common way of validating is to use both these methods at different times. Usually, hold-out validation is used in two phases of model creation. First, it is usually used while designing, to get a quick idea of the strength of the model. Second, hold-out validation is commonly used to avoid the inherent bias of training on all the data during the creation of the system. To avoid this, carefully designed hold-out validation can be used to have some completely unseen data points to test the model on at the very end. As you may have surmised, cross-validation is used in between these two, and also when doing a final comparison of different models.

2.6 Feature Vector

To input data to a machine learning model, the data that is being used has to be represented in some way. One way to do this is as an n-dimensional vector, called the *feature vector*, which represents a point in the *feature space*. A machine learning model is then being trained to map this input vector in some useful way, such as to a class or decision. Each element in this vector represents a single feature in the data. For example, for a binary feature vector, each element signifies whether this specific feature is present or not. For text classification problems, of which Native Language Identification is a subset, it is common for features to represent the presence or prevalence of terms in the text. For the machine learning methods to work well, it is very important for the input vector to be carefully designed, in order to make the data more easily separable in the feature space.

2.7 Statistical Machine Learning Methods

Since the system described in this thesis also experiments with the use of a Support Vector Machine meta-classifier, this method will be described in some detail. A quick run through of some other methods that are used by systems referenced in section 3 follows.

2.7.1 Support Vector Machines

As mentioned, Support Vector Machines (SVM) are very commonly used supervised machine learning models, introduced by Cortes and Vapnik 1995. An SVM is a linear classifier that finds a hyperplane that splits an n-dimensional feature space into two parts. Each data point is called a support vector, and the idea is that the best linear separation between two classes is found by keeping the distance between the hyperplane

and the closest support vector as large as possible. This distance is called the *margin*. A set of data points can be expressed as $(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)$, where y_i is either 1 or -1, depending on which class the ith item belongs to. Similarly, a hyperplane is given by

$$\vec{w}\vec{x} + b = 0 \tag{5}$$

where $\vec{w}$ is a weight vector for the feature space, and b is the bias. If the data is linearly separable, we can define two parallel hyperplanes boundaries where all items on or above $\vec{w}\vec{x} + b = 1$ are of one class, and all items on or below $\vec{w}\vec{x} + b = -1$ are of the other. The size of this margin is then given by $\frac{2}{||\vec{w}||}$, and to maximize this distance, we need to minimize $||\vec{w}||$, subject to $y_i(\vec{w}\vec{x} + b) \geq 1$.

This version is called hard-margin SVM, and was the method originally introduced, but this only works on linearly separable data. Whenever a point is found that is closer than the margin, the margin is made smaller. But a lot of data is not linearly separable, but we would still like a reasonable attempt. A second version, called soft-margin, was introduced to achieve this, and works by introducing a *hinge loss* function, which penalizes having data points on the wrong side, which is weighed against the value of having a wider margin. This means that the classifier is able to learn a model even if the data is not separable by a hyperplane.

Another way to separate data that is not linearly separable while still using Support Vector Machines, is to somehow make the data linearly separable, or at least more separable. Manipulating the data in this way is called the *kernel trick*, and involves feeding the data points into a *kernel function* that maps the data into another feature space where the data is linearly separable. This will often involve mapping into a higher dimension, for example through the use of the polynomial function. Working in a higher dimensional feature space does increase the generalization error, but using enough examples alleviates this problem somewhat.

Splitting a feature space between more than two classes is also something that the traditional SVM approach is unable to do. In order to do this, the problem is usually reduced to several binary classifications. The most common ways are creating an SVM model for each class and classifying opposed to the rest (one-vs-all), or to create a model for each pair of classes (one-vs-one). One-vs-all chooses the classification of the classifier with the highest degree of certainty as its final classification. On the other hand, when using one-vs-one a classifier is trained for each pair of classes, and each of the resulting models gives a vote to one of the two classes it has been trained on based on its classification. The class with the most votes is given as the final classification for the one-vs-one multiclass classifier.

2.7.2 Other statistical methods

Two other statistical methods have been used by important systems in NLI: Maximum Entropy and Linear Discriminant Analysis (LDA). A maximum entropy learner builds on the idea that the model which best represents given data, is the model with the highest amount of average information for an observation. Unlike SVM, the Maximum Entropy

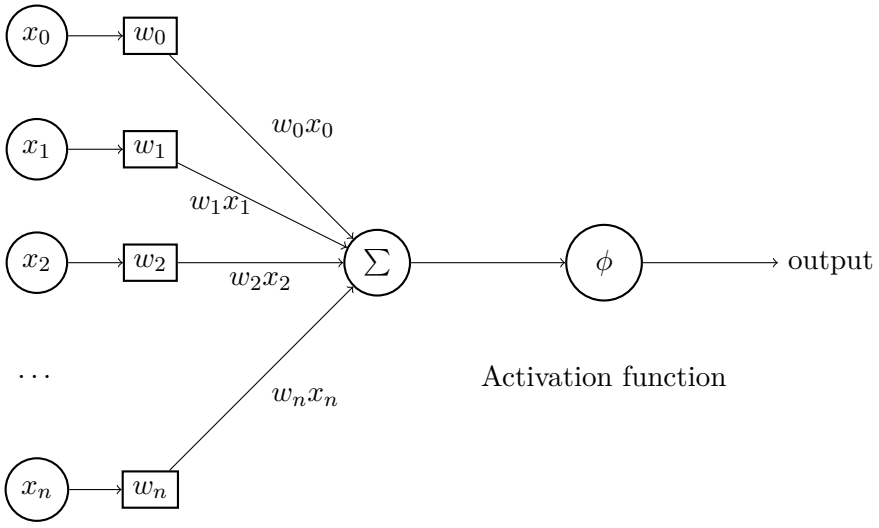

Figure 1: Model of a perceptron.

classifier model is not binary, although a Logistic Regression classifier is essentially a binary version of a Maximum Entropy classifier. Maximum Entropy learners have been quite commonly used methods within Natural Language Processing.

Linear Discriminant Analysis is a method that maps high dimensional data which is split into k classes down into a $(k-1)$-dimensional space. The goal is for the data to have the highest possible degree of separability between classes and least amount of separability within each class when mapped to this new space. More formally, it tries to find the linear combination of predictors that best separates the data.

2.8 Neural Networks

Since the main classifier used by the system which is the topic of this thesis is a shallow feedforward neural network, this section will go more into detail on the basics of neural networks, and only quickly summarize some more advanced neural network types that have been used in other NLI systems.

2.8.1 Basics

The most basic form of neural networks, the *perceptron*, was introduced by Rosenblatt (1958). It was inspired by the workings of a biological neuron, and consists of a set of weights that are multiplied with each of the perceptron's inputs, summed, added to a bias variable, and sent through the perceptron's *activation function*, which decides the output of the perceptron. Originally, this was a step function, where the perceptron

output was given by the function

$$\phi(x) = \begin{cases} 0 & \sum_{i=0}^{n} W_i x_i + b \leq 0 \\ 1 & \sum_{i=0}^{n} W_i x_i + b > 0 \end{cases} \tag{6}$$

Since this can lead to a complete change in output after only a small change in the weights, with very little opportunity to see how certain the perceptron is in its classification, the step function is usually no longer used. Ideally, a minor change in the input to a perceptron should result in a minor change in the output from that perceptron. Commonly used activation functions include sigmoid functions, the softmax function, and the rectified linear unit (ReLU) (see 2.8.3). The output of the perceptron separates the input into two classes (1 or 0), but due to the fact that it is based on a linear combination of weights and inputs, the separation is a hyperplane in the feature space, and, just like SVMs, it can not separate data that is not linearly separable. The learning method works by updating the weights after each step by the error of the output, modified by a learning factor. This is shown in the following equation

$$W_i(t+1) = W_i(t) + \alpha(d - y(t))x_i \tag{7}$$

where W_i is the ith weight, t is the current time step, d is the desired output, y is the actual output, and x_i is the ith input. This idea of the perceptron is the basis for more advanced neural networks.

As mentioned above, a single perceptron is only able to linearly split an n-dimensional space into two classes, since it is a linear combination of weights on the input with some activation function. Using a perceptron for multiple classes is simply a case of creating one perceptron for each class and choosing the class of the perceptron that fires most strongly on a given input. To avoid the linearity, it is necessary to add more layers. This type of neural network is known as a *multilayer perceptron* (MLP), or *feedforward neural network*, and all perceptrons in a layer take as their input the outputs from all neurons in the preceding layer.

One thing to note, is that unlike statistical classifiers like SVMs, neural networks will not necessarily find the global optimum on a given training session, since the way they are trained is by repeatedly looking at different subsets of the training data. However, when finding a good kernel function to make data easily separable is difficult, it can often be easier to train a neural network on that data.

2.8.2 Backpropagation

It is clearly much more difficult to train a network with several layers, since it is not immediately obvious how each weight in the network impacts the final output. The solution to this problem is an algorithm known as *backpropagation*, introduced by Rumelhart, Hinton, and Williams (1986). They showed that backpropagation was significantly faster than other methods for finding the gradient of the loss function. Backpropagation is the neural network version of *reverse-mode differentiation*, which is partial differentiation backwards through a computational graph. The introduction of backpropagation made

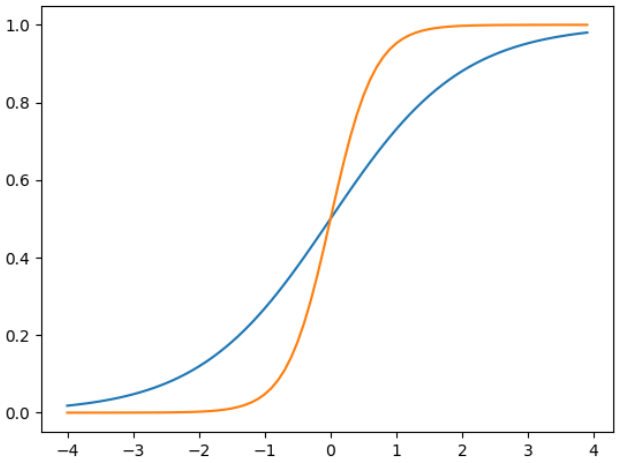

Figure 2: The logistic function is a sigmoid function, and is a commonly used activation function in artificial neural networks. Shown are two logistic functions using $k = 1$ and $k = 3$

.

neural networks able to solve problems that were previously considered computationally infeasible, and it is the central concept of training neural networks.

2.8.3 Activation functions

As mentioned in 2.8.1, the output from a perceptron is decided by that perceptron's *activation function*. Initially this was the step function, but due to the fact that small changes in input could lead to massive changes in output, alternative functions are now used. These are generally sigmoid functions, most commonly the logistic function (confusingly also commonly known as the sigmoid function). When using the logistic function, a perceptron is essentially a special case of Logistic Regression, and the version used as an activation function in artificial neural networks is given by

$$f(x) = \frac{1}{1 + e^{-kx}} \tag{8}$$

where k gives the steepness of the curve.

When looking at the graph of a sigmoid function, it is obvious that it bears a very close resemblance to the step function, with a fairly rapid flattening out as the sum of inputs gets farther away from 0. While this makes it more similar to biological neurons, it leads to a problem in perceptron networks when perceptron weights lead the input away from the center, since the gradient becomes less and less steep. Known as gradient saturation, this extremely shallow gradient leads to further learning becoming very slow, and for multilayer perceptron networks this shallow gradient also leads to

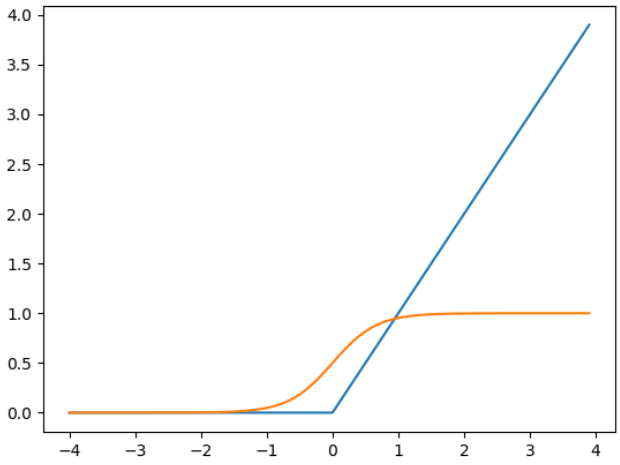

Figure 3: ReLU activation function compared to the logistic function with k=3

problems further back, since the perceptron will no longer propagate a gradient back through the network. Fairly recently, an alternative activation function that avoids this problem was introduced, the Rectified Linear Unit (ReLU) (Nair and Hinton 2010). ReLU is defined as

$$f(x) = x^+ = max(0, x) \tag{9}$$

As long as the final input is positive, there is always a constant gradient. Using ReLU does however have one problem. If the bias term of a perceptron becomes a large negative number that drowns out the summed weight input such that no input makes it fire, it becomes a dead neuron, and learning is completely impossible for that neuron, unlike for the sigmoid functions where there might still be some slight learning possible, however slow. Because of this, it is important to try to avoid negative bias terms, if possible, and the bias terms for a perceptron layer using ReLU activation are usually initiated to some small positive constant.

2.8.4 Other neural network designs

Two other very different neural network design types have seen significant use in Natural Language Processing generally, and have also been tried more recently in NLI: *Recurrent neural networks* and *convolutional neural networks*. A recurrent neural network (RNN) is a network where instead of sending its output forward in a network, some data is also sent back into the same layer in the next step. This means that an RNN can have a dynamic internal state (somewhat like a memory), and such networks are good at learning temporal connections such as that found in natural language. They are therefore commonly used for machine translation and speech recognition. The backpropagation for such networks require calculating the gradient at all previous steps. While traditional

RNNs have been good at learning short term dependencies, long term dependencies were elusive due to a problem of exploding and vanishing gradients (Pascanu, Mikolov, and Bengio 2013). Some RNN network types have been created to mitigate this problem, of which the most commonly used is Long Short Term Memory (Hochreiter and Schmidhuber 1997).

Convolutional neural networks (ConvNets) were originally created within the field of computer vision, where the method has been very successful at training computers to recognize objects, faces, etc. The idea behind ConvNets is that each neuron acts as a filter over a vector or matrix of inputs. In each layer a certain amount (usually a lot fewer than in traditional networks) of neurons pass over the preceding layer and filters it into a new output. This is easiest to explain with image recognition. On the 2-dimensional input image, each neuron passes over it and creates an output for each pixel based on some size of filter (3x3 or 5x5 is common). This is then saved as a new image as that neuron's output. However, since each neuron creates a full output, the amount of pixels is usually at least halved though a pooling function, since the amount of memory required would otherwise expand very rapidly with each layer. At the next level each neuron passes over all the output matrices from the preceding layer and creates a single new output layer of its own. Finally, after the last convolution layer, the output is flattened and one or more fully connected layers are added at the end to create the final classification.

For text, the process is very similar, except the representation of text is less obvious than it is for an image. Two main ways exist to do this. The traditional way is to add a word embedding layer at the beginning, trained as part of the network but often initialized to a pretrained layer to shorten training times. An embedding layer creates a vector for each word in the text. These vectors are then placed next to each other and the convolution is performed over the resulting matrix with a region size in one direction equal to the vector length and any desired size in the other (Y. Zhang and Wallace 2015). More recently, character level convolutional networks have also been used effectively (X. Zhang, Zhao, and LeCun 2015).

2.9 Natural Language Processing

Natural language processing (NLP) is the study of helping computers parse and understand natural languages. Terms and concepts related to NLP that are used a lot in this thesis are explained here.

2.9.1 Syntax and semantics

The syntax of a language is the set of rules and processes that define how to create valid statements in that language. However, it does not give any information about whether the sentence makes any sense or what it means. It is mainly about the order of tokens in statements, and is a subset of grammar. Contrasting this, semantics focuses on signifiers, such as words or phrases, and the relationship between them, and is the study of meaning in languages.

The	quick	brown	dog	jumps	over	the	lazy	fox	.
DT	JJ	JJ	NN	VBZ	IN	DT	JJ	NN	.

Figure 4: Simple sentence tagged by the Stanford tagger.

2.9.2 Part-of-speech

Part-of-speech (POS) is a lexical categorization of words where the words are given a class based on their grammatical properties. These are classes like noun, verb, adjective, etc. To avoid tagging texts manually, a POS tagger has to be created and trained for a specific language. Since this is a lot of work, pre-created taggers are ususally used. For English, the most common tagger is the Stanford POS Tagger (Toutanova et al. 2003), which uses the Penn Treebank English POS tag set, which includes 36 tags, including general ones such as *Verb, past tense* (VBD) and *Proper Noun, singular* (NNP), and more particular ones, like *Wh-pronoun* (WP), *List item marker* (LS), and *to* (TO). Figure 4 shows a simple sentence and its associated POS tags. The Stanford Tagger is used by the system outlined in this thesis.

2.9.3 N-grams

An n-gram is a substring of length n of a string of tokens. These tokens can be anything, but the most common in the context of NLP are character, word, and POS. As an example, for the sentence "The quick dog jumps over the lazy fox", some n-grams include *brown fox jumps* (word trigram), *JJ NN VBZ* (POS trigram), and *mps ov* (character 6-gram). A look at some complete n-gram sets from the sentence in figure 4 can be found in table 1.

2.9.4 Bag-of-words

Bag-of-words is a simple and common way of representing text for text classification tasks. In a bag-of-words vector, each element in the vector represents some term in the text. For each text, the terms are either counted or, for a binary vector, marked as present or not. The order in which the terms are found in the text is not represented, although the model can be used with any terms, such as n-grams to show ordering, part-of-speech tags to represent grammar, etc. A bag-of-words model of two sentences can be seen in table 2.

2.9.5 Term frequency - inverse document frequency

In order to help decide which terms are important to a document in relation to its corpus, it is fairly intuitive to look at terms that are rare in the overall corpus but used in some documents. These can then be weighted to make them have more of an impact on the position of those documents in the feature space. This is the idea behind term frequency - inverse document frequency (tf-idf).

Type	Size	Set
Character	trigrams (3)	The, he␣, e␣q, ␣qu, qui, uic, ick, ck␣, k␣b, ␣br, bro, row, own, wn␣, n␣d, ␣og, dog, og␣, g␣j, ␣ju, jum, ump, mps, ps␣, s␣o, ␣ov, ove, ver, er␣, r␣t, ␣th, the, he␣, e␣l, ␣la, laz, azy, zy␣, y␣f, ␣fo, fox, ox␣, x␣.
	4-grams	The␣, he␣q, e␣qu,␣qui, quic, uick, ick␣, ck␣b, k␣br, ␣bro, brow, rown, own␣, wn␣j, n␣ju, ␣jum, jump, umps, mps␣, ps␣o, s␣ov, ␣ove, over, ver␣, er␣t, r␣th, ␣the, the␣, he␣l, e␣la, ␣laz, lazy, azy␣, zy␣f, y␣fo, ␣fox, fox␣, ox␣.
Word	unigrams (1)	The, quick, brown, dog, jumps, over, the, lazy, fox, .
	bigrams (2)	The quick, quick brown, brown dog, dog jumps, jumps over, over the, the lazy, lazy fox, fox .
POS	bigrams (2)	DT JJ, JJ JJ, JJ NN, NN VBZ, VBZ IN, IN DT, DT JJ, JJ NN, NN .

Table 1: Some n-gram sets in the sentence from figure 4. Note that the commas in the character n-gram lists are the delimiters, and that spaces are indicated by ␣.

Terms	Document 1	Document 2
the	2	2
quick	1	0
brown	1	0
dog	1	1
jumps	1	0
over	1	0
lazy	1	0
fox	1	0
.	1	1
bowl	0	1
belongs	0	1
to	0	1

Table 2: Bag-of-words for a corpus of two short documents, one of which is the sentence from figure 4.

Term frequency (tf) is a measure of how common a given term is in the current document. The two most basic ways of looking at a term's frequency in a document is raw count and binary. Raw count, as the name implies counts the amount of times a term is found in the document, while binary only checks whether a term is present or not. Since the raw count, denoted by $f_{t,d}$, is prone to weighting longer documents more highly than short ones, it is usually scaled, either by simply dividing by the total amount of terms in the document, or by dividing all the terms by the highest $f_{t,d}$ in the document. The last one is the most common and is given by

$$\text{tf}(t,d) = K + (1-K)\frac{f_{t,d}}{\max_{t' \in d} f_{t',d}} \tag{10}$$

where K is a *smoothing term* between 0 and 1 (0.4 and 0.5 are common).

Inverse document frequency (idf) is a measure of how rare a term is in the entire corpus and is the logarithmically scaled fraction of documents in the corpus that contain the term.

$$\text{idf}(t,D) = \log\frac{N}{|\{d \in D : t \in d\}|} \tag{11}$$

where D is the corpus and N is the total amount of documents in D. For a given document, the tf-idf score of each of its terms is found by multiplying that term's tf and idf scores.

$$\text{tf-idf}(t,d,D) = \text{tf}(t,d) \cdot \text{idf}(t,D) \tag{12}$$

As an example, let us look at the terms *the* and *brown* in the small corpus from table 2. First we find the term frequency for each term in each document and inverse document frequency of each term in the corpus.

$$\begin{aligned}
\text{tf}(\text{“the”}, d_1) &= 0.4 + 0.6 \cdot \frac{2}{2} && = 1.0 \\
\text{tf}(\text{“the”}, d_2) && = 1.0 \\
\text{tf}(\text{“brown”}, d_1) &= 0.4 + 0.6 \cdot \frac{1}{2} && = 0.7 \\
\text{tf}(\text{“brown”}, d_2) &= 0.4 + 0.6 \cdot \frac{0}{2} && = 0.4 \\
\text{idf}(\text{“the”}, D) &= \log\frac{2}{2} && = 0 \\
\text{idf}(\text{“brown”}, D) &= \log\frac{2}{1} && \approx 0.301
\end{aligned}$$

As expected, the term *the*, while very common in both documents, is removed by tf-idf, since the term gives less obvious information about the documents' place in the corpus. On the other hand, *brown* will be kept, since it is only present in half of the corpus. The final tf-idf scores of *brown* are given by

$$\begin{aligned}
\text{tf-idf}(\text{“brown”}, d_1) &= 0.7 \cdot \log(2) && \approx 0.211 \\
\text{tf-idf}(\text{“brown”}, d_2) &= 0.4 \cdot \log(2) && \approx 0.120
\end{aligned}$$

2.10 Tools

This section will go through the tools used by the system in this thesis. In addition to the tools mentioned here, the Stanford Tagger (Toutanova et al. 2003), also referenced in 2.9.2, was used for POS tagging of all the documents in the dataset.

2.10.1 NLTK

NLTK (Bird, Klein, and Loper 2009) is a leading open source set of libraries for Natural Language Processing using Python. It contains many useful methods for collocations, tokenizers, grammars, automatic calculation of probabilistic distributions, parse trees, etc. In the system for this thesis, it is only used for creating the n-grams, and to have a Python interface to the Stanford Tagger, which is written in Java.

2.10.2 TensorFlow

TensorFlow (Abadi et al. 2015) is an open source library developed by Google for numerical computations in flow graphs, for example to create neural networks. Using TensorFlow involves first builing a flow graph, and then using one of the many training functions on the end result. For the system outlined in this thesis, it is used to create the flow graph for the shallow neural network classifiers, and training them.

2.10.3 Scikit-learn

Scikit-learn (Pedregosa et al. 2011) is a machine learning library for Python. It contains implementations of many common machine learning methods, such as Logistic Regression, Support Vector Machines, k-nearest neighbor, etc. For the system in this thesis, the linear SVM classifier is implemented using scikit-learn's LinearSVC, which is an interface to LIBLINEAR's (Fan et al. 2008) implementation of a linear SVM classifier. Also, tf-idf weighting and chi squared feature selection is implemented using scikit-learn, as well as calculating of precision, recall, and F-measures, and creating the confusion matrices.

3 Related Work

As mentioned in the introduction, research into NLI by computer scientists only took off at the end of the previous decade. This research can be split into two main phases. First, the early phase, which focused on researching baseline systems and corpora. This phase culminated in a large shared task in 2013, where 27 teams compared results on the same dataset. Second, the period after the shared task, where the results from the shared task were explored further and improved upon, as well as some smaller explorations, including a shared task for NLI using speech only. This second phase culminated in another large shared task using both text and speech in 2017.

3.1 Early research

While not the first work in the field, Koppel, Schler, and Zigdon (2005) created the first baseline system for this task. Using the International Corpus of Learner English (ICLE) (Granger et al. 2002) as their corpus, and looking at a set of 1035 features, they achieved an accuracy rating of more than 80%. They used work from the field of authorship attribution, and modified it to identify the class (native language, L1) the author belonged to, rather than looking for a specific author. In their system, they used four feature types (function words, character n-grams, error types, and POS n-grams) all of which have been used to some extent in most later systems for NLI. The size of the feature space used (1035 features in total) was, however, quite small compared to later work, which can be seen quite clearly when we look at the submissions to the first shared task. In Tsur and Rappoport (2007) they found that even using just character n-grams worked very well, although the results, both from Tsur and Rappoport (2007) and Koppel, Schler, and Zigdon (2005), may have been influenced by the topic bias found in ICLE, leading to the learning of topic words rather than words unique to speakers of the same L1. This bias was first brought up by Brooke and Hirst (2013), who found significant drops in accuracy when doing tests accounting for topic bias, even in features that were previously assumed to be immune to such biases, like POS n-grams and function words.

Noting the lack of use of syntactic features in NLI, Wong and Dras (2011) looked at using parse structures to create feature sets on the ICLE. Looking at the previously used lexical features as a baseline, they created a new semantic feature type based on parse trees. They took horizontal slices from parse trees and treated them as Context-Free Grammar (CFG) production rules, and then created binary feature vectors for each text (i.e. a text either contained a given production rule or not). After training a Maximum Entropy machine learner to act as the classifier, they reported an accuracy of 80% on the ICLE with hold-out validation, and 77.75% accuracy with 5-fold cross validation.

Based on this use of Context-Free Grammars to create feature sets for NLI, Tree Substitution Grammars (TSGs) were brought up as an alternative by Swanson and Charniak (2012), where they achieved an accuracy of 78.4% on the ICLE with their best model. They experimented with two different heuristic types and two ways of looking at the data (sentence or complete document), and compared this to the performance of

using Context-Free Grammars. While Swanson and Charniak do not report what kind of validation was used for their result, they do compare their results to those of Wong and Dras. Swanson and Charniak report that they implemented the CFG as described in Wong and Dras (2011), and this CFG model gave a worse accuracy than all three TSG implementations when looking at the whole document, and worse than the best system when looking at sentences individually and voting.

Tetreault, Blanchard, Cahill, and Chodorow (2012) reached a new peak on the ICLE corpus, at 90.1% accuracy, but they expanded on the problems of ICLE for NLI. In addition to the topic bias, there are also encoding errors that vary from language to language. These together lead to learners learning specific topic words, or the encoding, rather than language mistakes tied to an L1. To address these problems, Tetreault, Blanchard, Cahill and Chodorow proposed two solutions, first by changing the ICLE corpus they worked on, creating a subset called ICLE-NLI. Second, they introduced a new corpus made specifically for NLI, TOEFL11 (Blanchard et al. 2013), which has since become the de facto standard corpus for NLI, especially since its use in the 2013 Shared Task. Since this is also the dataset that is used for the experiments in this thesis, a description of the dataset follows.

3.2 The TOEFL11 Dataset

The original TOEFL11 data set consisted of 12,100 English language essays from the Test of English as a Foreign Language (TOEFL), evenly split between 11 different native languages (L1s): Arabic, Chinese, French, German, Hindi, Italian, Japanese, Korean, Spanish, Telugu, and Turkish. This means each language has 1,100 essays. These essays were written as answers to 8 different questions (henceforth called *prompts*). An attempt has been made to make them as evenly distributed between the 8 essay prompts as possible, but there is a certain amount of difference in distribution for the different L1s, especially with regard to prompt 6, where several of the languages have very few essays. In fact, only 4 languages (Arabic, Chinese, Japanese, and Korean) have more than 100 essays for all prompts. A complete overview of the distributions can be seen in table 3. For each of the shared tasks 1,100 more essays were added to the set as hold-out validation sets, raising the amount up to 13,200 essays. Note that the prompt identification numbers differ between the original dataset and the version given out in 2017. Both of the prompt identifiers are shown in table 3.

In the original data set, proficiency levels (based on the essay writer's result in the TOEFL) for each essay were also provided, but since the inclusion of proficiency did not give noticeable improvements to the accuracy of classifiers, this was omitted in the set for the 2017 Shared Task.

3.2.1 Text data

The essay data is released in two different formats, raw and tokenized. The raw data is the set of essays from the TOEFL without any preprocessing. As such, this is the dataset to use when exploring different tokenizing and early preprocessing steps. The to-

Language	Prompts							
	1	2	3	4	5	6	7	8
	P4	P7	P6	P0	P5	P1	P3	P2
Arabic	138	137	138	139	136	133	138	141
Chinese	140	141	126	140	134	141	139	139
French	158	160	87	156	160	68	151	160
German	155	154	157	151	150	28	152	153
Hindi	161	162	163	86	156	53	158	161
Italian	173	89	138	187	187	12	173	141
Japanese	116	142	140	138	138	142	141	143
Korean	140	133	136	128	137	142	141	143
Spanish	141	133	54	159	134	157	160	162
Telugu	165	166	167	55	169	41	166	171
Turkish	169	145	90	170	147	43	167	169
Total	1,656	1,562	1,396	1,509	1,648	960	1,686	1,683

Table 3: Amount of essays for each L1 and prompt. Numbers are taken from Blanchard et al. (2013), and therefore only includes the 12,100 essays that are publically available. The first row of prompt numbers are the ones found in the original data set, while the second row are the prompt identification from the 2017 Shared Task.

kenized set consists of the essays pre-tokenized. That is, all tokens (such as punctuations, contractions, and quotation marks) are separated by a space.

As an example, let us look at a partial sentence from the dataset, more specifically the test taker that was given the identification number 7173 in the 2017 Shared Task version of the dataset. In the raw data, the example partial sentence is written "In my life I don't like lies, and I don't like liar people, (...)". In contrast, the tokenized text is written "In my life I do n't like lies , and I do n't like liar people , (...)". Note the space between the commas and the preceding word, and the split of *don't* into *do* and *n't*.

There is a lot of variance in the length of essays in the data set. The shortest essay is only two words ("I agree."), while the longest is 876 words, with an average length of approximately 300 words. Obviously, short essays are much more difficult to classify than longer ones, but at the same time, the long ones are usually correlated with high proficiency, and one could imagine that some obvious mistakes, such as spelling mistakes might be much more rare for these essays.

3.2.2 Speech data

The speech data was added to the dataset for the 2017 Shared Task. As part of the TOEFL, participants' responses to speech prompts were originally recorded, and these had been kept. Ideally, these raw recordings should be available for classification, but for the privacy of the test takers, it was not possible to make such a wide release of this data. Two versions of the speech data was released: *transcripts* and *i-vectors*. The transcripts

were manually created orthographic transcriptions of lengths between 0 and 202 words. Since the i-vectors are used as is by the papers in the 2017 Shared Task and this work, it is sufficient to know that an i-vector is a vector of fixed length (in this case 800), which has been created as a lower-dimensional representation of high-dimensional sequential recordings of speech data. They were originally created for use in speaker recognition (Dehak et al. 2011), later used for language recognition (Martinez et al. 2011), and used at the paralinguistics challenge at Interspeech 2016 for NLI (Senoussaoui et al. 2016).

3.3 2013 NLI Shared Task

In 2013, the first shared task for NLI was held to have a better comparison between different NLI systems. Using the TOEFL11 data set and adding another 1,100 essays as the hold out validation set, 29 teams submitted solutions to the task, with accuracy scores ranging from 84.6% to 63% for the 13 teams that did 10-fold cross validation on the full set after the task had finished. A longer summary can be found in Tetreault, Blanchard, and Cahill (2013), but a short summary of the systems follows here.

Jarvis, Bestgen, and Pepper (2013) reported the best results for the initial holdout validation, at 83.6% on the test set, and second best after 10-fold cross validation, at 84.5%. They used a bag-of-words feature vector consisting of lexeme, lemma, and POS 1-, 2-, and 3-grams that were present in at least two texts in the training set. This vector was normalized to unit length, and a log-entropy weighting was applied, both of which gave improved accuracy. They also optimized the cost parameter of their SVM. Since none of the features used were new, and they did not employ a comparatively large feature vector, these optimization and weighting methods are likely to be a major reason for their good results.

Motivated by the possibility that simple lexical features could efficiently be used on the task without the need to learn syntactic and morphological differences between L1s, Lynum (2013) trained a system using only lexical features. As features he used word unigrams and bigrams, character n-grams, with n ranging from 3-6 and 1-7 depending on the system, and suffix bigrams. All these features were weighted through tf-idf weighting. As most systems in the shared task, a linear SVM classifier was used on these feature vectors, and like Jarvis, Bestgen, and Pepper (2013), the cost parameter was optimized with 5-fold cross validation. This system reported the second best results, at 83.4%.

One thing to note is the huge feature space found in a lot of the solutions. The best systems all used large n-grams (up to 9-gram for characters for one system), but some systems made the feature vectors smaller by omitting features that did not occur frequently, or that occurred often enough that they were less useful for distinguishing between L1s. For example, Gebre et al. (2013) omitted features that occurred less than 5 times or that occurred in more than 50% of the essays to reduce the feature space to 73,626, which is still significantly higher than the small feature space from Koppel, Schler, and Zigdon (2005). Other systems (Jarvis, Bestgen, and Pepper 2013; Lynum 2013; Popescu and Ionescu 2013) had much larger feature spaces, most notably Popescu and Ionescu (2013) with a total feature space size of 7,669,684.

Having such large feature spaces necessarily means that the learning algorithms need

Rank	Team	Classifier	Feature space	Accuracy	10-fold CV
1	Jarvis, Bestgen, and Pepper (2013)	SVM	> 400,000	83.6%	84.5%
2	Lynum (2013)	SVM	867,479	83.4%	83.9%
3	Popescu and Ionescu (2013)	KRR	7,669,684	82.7%[1]	82.6%
6	Goutte, Léger, and Carpuat (2013)	SVM Ensemble		81.8%	
7	Tsvetkov et al. (2013)	MaxEnt	5,664,461	81.5%	
8	Gebre et al. (2013)	SVM	73,626	81.4%	84.6%[2]
13	Malmasi, Wong, and Dras (2013)	SVM Ensemble		80.1%	82.5%

[1] : 82.5% without heuristics based on knowledge of the test set.
[2] : When doing 10-fold cross validation, this system did better than all other systems.

Table 4: An overview of some key systems in the 2013 Shared Task.

to be relatively simple in order to remain computationally feasible. Thirteen of the submitted systems used SVM classifiers, and four of those consisted of several SVM classifiers set up in an ensemble. For example, Goutte, Léger, and Carpuat (2013) found that classifiers with very similar accuracy could disagree on a large percentage of the dataset. This led them to using a simple majority vote to take advantage of combining classifiers trained on different features (character n-grams, PoS n-grams, etc.). Malmasi, Wong, and Dras (2013) experimented on different combination methods for SVM classifiers and found that, for their features, using the sum combination method worked best. They also concluded that the differences were small and that any new ensemble combination method would have to be very different from existing combination methods to expect greater improvement on the results.

Only two other approaches were used in systems that achieved more than 80% accuracy, Logistic Regression and String Kernels. Logistic Regression was used in Tsvetkov et al. (2013) to achieve 81.5% accuracy using a feature set of 5,664,461, a set that is more than ten times larger than that of the best system. This feature set included a large amount of non-standard features, a lot of which had a fairly low impact on system accuracy.

The string kernel method proposed by Popescu and Ionescu (2013) consists of a one-vs-all Kernel Ridge Regression classifier system using two kernel methods, String Kernels and Local Rank Distance. Their String Kernel simply looks at how many substrings of a given length are shared between two texts, either in a binary fashion, where a substring is either shared or not, or counted for each text. Both of these are usually normalized. Local Rank Distance measures the offset of similar character n-grams within a string, in this case the whole document, with some maximum offset. This maximum offset is used to avoid searching too far, as well as making sure normalization is simple. Using all of this together, Popescu and Ionescu achieved an accuracy of 82.5% for their best system.

They also used a set that exploited knowledge about the testing set to create a heuristic to improve accuracy, which performed slightly better, at 82.7%, but this does not show the actual performance of the system itself.

3.4 Between the Shared Tasks

Kríž, Holub, and Pecina (2015) took ideas from other parts of NLP to greatly reduce the feature space while keeping comparable accuracy to the better systems in the 2013 Shared Task. They based this system on two new ideas for NLI, namely Language Modeling and looking at the difference in cross-entropy scores between the text and the language models. The scores were calculated for each feature family (tokens, characters, suffixes, and POS tags) and these scores were then used as features for a linear SVM classifier, which as in previous research was found to be no worse than more advanced classifiers. In addition to the 44 cross entropy features, 11 more features were added. Two of these features, Prompt and Proficiency, were tags for each text and are not useful for data that unlike TOEFL11 does not have such tags. With only these 55 features, they reported an accuracy of 82.4%, which was competitive with the best systems from the preceding shared task.

Malmasi and Dras (2017) built on the results from the 2013 Shared Task and explored different ensemble methods and ensemble stacking, a method where a second ensemble is trained on all the outputs from the classifiers of the first ensemble. Testing was done on three different corpora, and they used a large amount of different classifiers and features. This ensemble stacking with Linear Discriminant Analysis classifiers reached a new best result on the TOEFL11, at 87.1% accuracy on the test set, and even using only a single LDA meta-classifier rather than an ensemble reached 86.8%. They also brought up the idea of Statistical Significance to NLI, since the reported accuracy is now becoming very close to the oracle baselines, and that it could be very beneficial for comparing systems when the differences in accuracy become smaller. Malmasi and Dras (2017) proposed using McNemar's test, due to its wide use for pairwise classifier comparison, and it was noted that comparisons with older systems was difficult due to the fact that the actual predictions of the systems are rarely released. Furthermore, in order to improve comparisons as systems approach the oracle upper bounds, they suggested that future systems should release this data, and showed their significance testing by comparing their new system to two of the best performing systems from the 2013 Shared Task.

At the Interspeech conference, a Computational Paralinguistics challenge is set every year, and in 2016, the challenge was related to NLI. The task, called "Deception, Sincerity & Native Language" (Schuller et al. 2016), had as one of its subtasks the challenge of identifying speakers' native language based only on an audio-recording of them speaking in English. The main difference between these systems and the text based systems is in the features used for training. When using sound for identification, phonetics, acoustics and phonotactics become important features, currently more so than lexical and syntactic content.

The best system for the Native Language subtask, by Abad et al. (2016), used Phone Log Likelihood Ratios to find acoustic-phonetic features, which were decoded through

Rank	Team	Classifier	Accuracy	F1
1	Cimino and Dell'Orletta (2017) Lab	Stacked SVM	88.18%	88.18%
1	Kulmizev et al. (2017)	Linear SVM	87.55%	87.56%
1	Goutte and Léger (2017)	Voting ensemble	87.36%	87.40%
1	Ionescu and Popescu (2017)		86.91%	86.95%
1	Li and Zou (2017)	Ensemble	86.55%	86.54%
2	Oh et al. (2017)	DNN ensemble	86.00%	86.01%
3	Bjerva et al. (2017)	Resnets+MLP+CBOW	83.18%	83.23%

Table 5: Overview of some essay-only track systems.

trained Multilayer Perceptron networks. These, along with some other acoustic and phonotactic features, were used for training with Total variability modelling, also known as the i-vector framework, which was also used by Senoussaoui et al. (2016). Abad et al. (2016) found that using ConvNets or Recurrent Neural Networks gave worse performance for the final training on these features. Their final system got an accuracy of above 80% (82.9% accuracy on the development set and a reported 81.3% Unweighted Average Recall on the test set[1]).

3.5 2017 NLI Shared Task

A second large shared task was held in 2017, combining the previously separate fields of text-based and speech-based NLI (Malmasi, Evanini, et al. 2017). While the text dataset was the same as the one from 2013 except a new test set, the speech part of the task consisted of previously unused manually created speech transcripts, and optionally a set of i-vector features created from the recorded speech of the TOEFL test takers. The i-vectors were added to give a more realistic idea of the performance of speech-based classifiers. Section 3.2 describes the dataset in more detail. The task was split into three tracks, essay-only, speech-only, and fusion. This section will look at the essay-only and fusion tracks, since the essay-only track had the most participants and the most in common with previous work, and the majority of teams that submitted systems to the speech-only track also submitted systems to the fusion track, whereas some teams with key systems from the essay-only track did not. Each task was also split into open and closed competitions, depending on what training data was used. The open competition allowed use of other training data in addition to the one provided by the organizers of the shared task. The organizers used statistical significance testing to place the teams into groups where the best and worst results were not significantly different. These groups were then ordered from best group to worst group, and all teams within a group were given the same *rank*.

Most of the top ranked essay-only systems used SVM classifiers in some configuration with different n-grams features; only a single rank 1 system (Li and Zou 2017) used neural

[1]No accuracy was reported for the test set.

Rank	Team	Classifier	Accuracy	F1
1	Ionescu and Popescu (2017)		93.18%	93.19%
1	Oh et al. (2017)	SVM/DNN Ensemble	92.18%	92.20%
1	Goutte and Léger (2017)	Voting ensemble	91.91%	91.93%

Table 6: Overview of some fusion track systems.

methods, and in that case only as the meta-classifier for a statistical ensemble. The best of these traditional systems was the one delivered by Cimino and Dell'Orletta (2017). They based their system on the idea that there are differences in the importance of features for sentence classification as opposed to document classification. Combining this idea with ensemble stacking, their system achieved an accuracy of 88.18% on the 2017 test set. The stacking consisted of an SVM sentence classifier, whose predictions were used as features by a second SVM document classifier. Both classifiers used most of the standard features, such character n-grams (up to 8-grams), word n-grams (up to 4), and POS n-grams (up to 4) as well as syntactic features, such as linear dependency and hierarchical dependency type n-grams. During experiments, they found that the syntactic features did not have a statistically significant effect on system performance.

Kulmizev et al. (2017) showed that very simple feature sets could still be used with good results. Their best system used only a binary feature vector of 1-9 character n-grams normalized using tf-idf, with an official test accuracy of 87.55%. They also looked at the prompt distributions for different languages in the data, and found some evidence of topic bias, which they tried to correct for by omitting topic words. When they also did experiments dropping prompts entirely from the training data, they found drops in accuracy when tested against that prompt. This shows that systems might have trouble generalizing toward unseen prompts, and they suggested that a true metric of a system can only be obtained by testing a system against such unseen prompts.

Continuing their work from the 2013 Shared Task, Goutte and Léger (2017) looked at the use of voting in ensemble classifiers. They found that the gains from using a voting ensemble were small, but consistently present given the same estimator, but with variability between estimators. Because of this, they concluded that there might still be room for improvement in voting systems. They also brought up the question of whether it is time to use long character n-grams in place of word n-grams now that computers can handle large n-grams. This choice of only using character n-grams would remove the need for linguistic preprocessing, even tokenization, and it might be able to model word stems without linguistic modeling.

Oh et al. (2017) delivered a deep learning based system using latent semantic analysis (LSA). Count based vectors from different length character and word n-grams went through an extensive preprocessing step to end up with fairly small feature vectors. Three deep neural network classifiers (DNN) took these features, DNN bottleneck values, and the i-vector as inputs, and the output of the three classifiers was fed into a final DNN meta-classifier. This performed very well, especially on the fusion track, where it achieved a top ranked accuracy score of 92.18%.

Bjerva et al. (2017) used a mix of Deep residual networks (resnet) directly on the text, a sentence level Bidirectional LSTM on POS tags, logistic regression on spelling features, and a continuous bag of words (CBOW) model. Their best system, which skipped the use of the LSTM, had a rank 3 accuracy of 83.18% in the essay-only track, but even using only resnets gave an accuracy of 80.27%.

The system that performed the best on the fusion track was delivered by Ionescu and Popescu (2017), who tried to see if a String Kernel system was still able to deliver state of the art results in NLI. Along with the string kernels for the text, they also developed a kernel based on the provided i-vectors. They experimented on using Kernel Discriminant Analysis instead of Kernel Ridge Regression, and found that it performed consistently better. Their best system had an accuracy of 92.09% on the fusion track and 86.91% on the essay-only track, both of which amount to rank 1 results. Their system shows that a system using only the character data can achieve top results.

4 System Design

The goals of this thesis is to explore the ability of a simple MLP classifier to get good results in NLI, and to look for an input preprocessing layer that is less prone to the drops in accuracy brought up by Kulmizev et al. (2017) when confronted with texts about an unseen topic. This section will outline the two proposed MLP designs, as well as the proposed preprocessing method, which will be compared to the more standard tf-idf / χ^2 selection which is commonly used in bag-of-words models.

4.1 Dataset and Preprocessing

The data used for the system is the TOEFL11 version from the 2017 shared task, without the test set. As such, it consists of essays and speech data from 12,100 test answers, split between 11 different native languages, as outlined in 3.2. It is split into two sets, a development set consisting of 100 essays, speech transcripts, and i-vectors from each L1, and a training set consisting of 1,000 from each L1. The essays are written as answers to 8 different prompts, while the spoken answers were given to 9 different questions.

- **Character n-grams** are extracted from the tokenized essays, varying in size from 3-9.
- **Word n-grams** are created from the tokenized essay data varying in size from 1-5.
- **POS n-grams**. The POS tags are found using the free Stanford Tagger (Toutanova et al. 2003) on the tokenized essay data, and 1-4-grams are created.
- **Speech data**. The provided i-vector data is used as the only speech data for the system. Transcripts will not be used, since the transcript data provided much less additional accuracy to systems in the 2017 Shared Task compared to the i-vectors.

Systems have had trouble generalizing to texts written for unseen prompts, and for this reason, part of the goal of this thesis is to look for preprocessing methods that could mitigate this drop in accuracy. To get a baseline result of the drop in accuracy, the preprocessing step will consist of using tf-idf weighted items selected through χ^2 feature selection. Both tf-idf and χ^2 were implemented using scikit-learn (Pedregosa et al. 2011). This method is chosen since it is a tried and true method for preprocessing text data into a bag-of-words model. However, one of the reasons this method could have drops in accuracy when faced with unseen prompts is that many of the important words chosen could possibly be chosen specifically due to the fact that they give very good information for specific prompts. For example, one could imagine that a prompt stating "Describe something that surprised you when coming to another country." could lead to writers using the name of their own country extensively in their text, an item which would be completely useless in prompts where a comparison between countries is not needed.

To combat this inherent bias toward high value items, a much simpler method of choosing n-grams is proposed. First, all items in the corpus are counted. Then, all

n-grams are sorted by how common they are, and the x most common items are chosen for the feature vector. These items are then linearly normalized within its vector. This simplistic choice will in all likelihood drop accuracy on known prompts, but hopefully this lower dependency on highly valued items could lead to a better ability to generalize. Another problem this method might have is that longer n-grams might be left out of the feature vector, since one could assume that long n-grams are more varied and therefore any given n-gram will be more rare. To combat this problem, experiments are also run looking at an individual feature vector for longer n-grams (n from 7-9), and comparing this to the result of using a single n-gram vector (n from 3-9).

4.2 Classifiers

Traditional statistical methods have been used a lot in NLI in the past, while neural methods have only recently been explored. The first goal of this thesis was to look at whether simple MLP classifiers could achieve good results for the task. To this end, this thesis is experimenting with two main classifier types. Simple MLP classifiers and MLP classifier ensembles. Since the ensembles essentially use simple MLP classifiers as their building blocks, the MLP classifiers will be described first.

The simple classifiers that are experimented with are two different types of fully connected perceptron networks. The most basic one is a linear classifier with one perceptron for each native language to be classified, each directly connected to each item in the input feature vector. This type of network is shown in figure 5. The second type is a multilayer perceptron, with the two-layer version shown in figure 6. The amount of layers could have increased further, but experiments described in section 5 found that even two layers completely fit the input data, and that more layers would not increase performance. The size of the hidden layer is also experimented on. The activation function used for all layers in the network is the Rectified Linear Unit (ReLU). The most basic classifier is therefore an MLP classifier that takes as its input the concatenated feature vectors of all the inputs.

The ensemble classifier, shown in figure 7, is designed using these MLP classifiers as its building blocks. First, individual MLP classifiers are trained for each of the features. Then, a final classifier is trained on the concatenated outputs from each of these individual classifiers. Two versions of this meta-classifier are experimented with. The first version is an identical MLP classifier, while the alternative is a one-vs-rest linear multiclass SVM meta-classifier implemented using LinearSVC from scikit-learn.

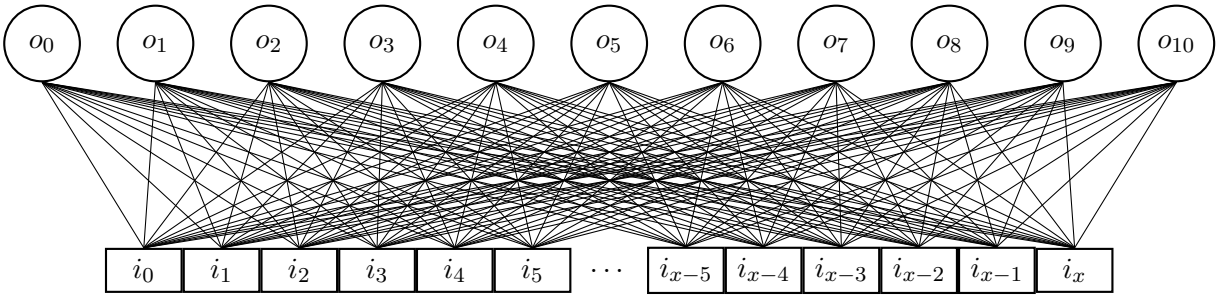

Figure 5: Direct input perceptron network with a single output node per native language.

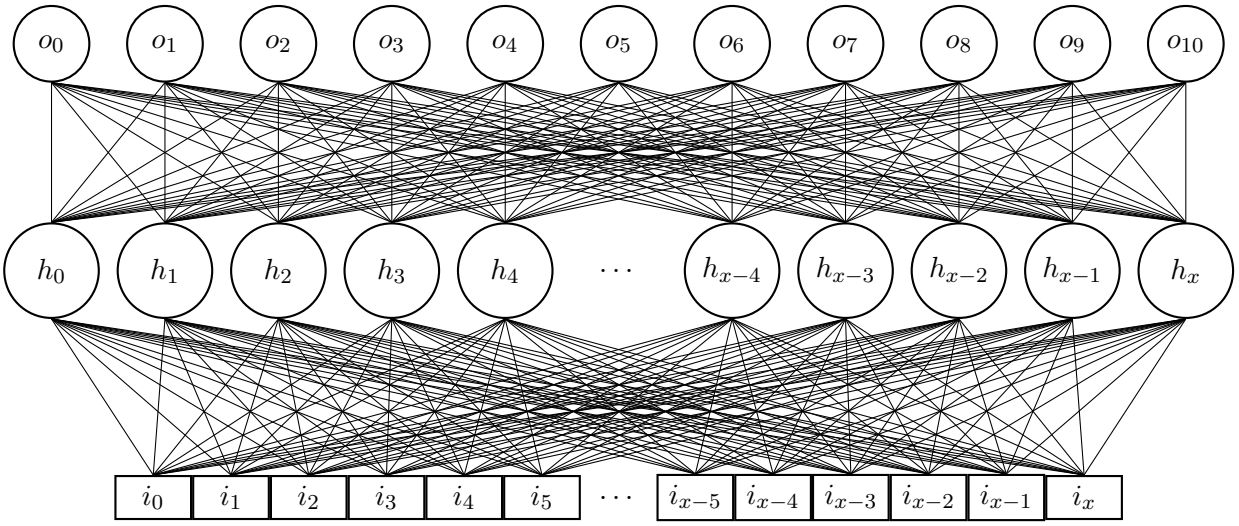

Figure 6: Two layer perceptron network.

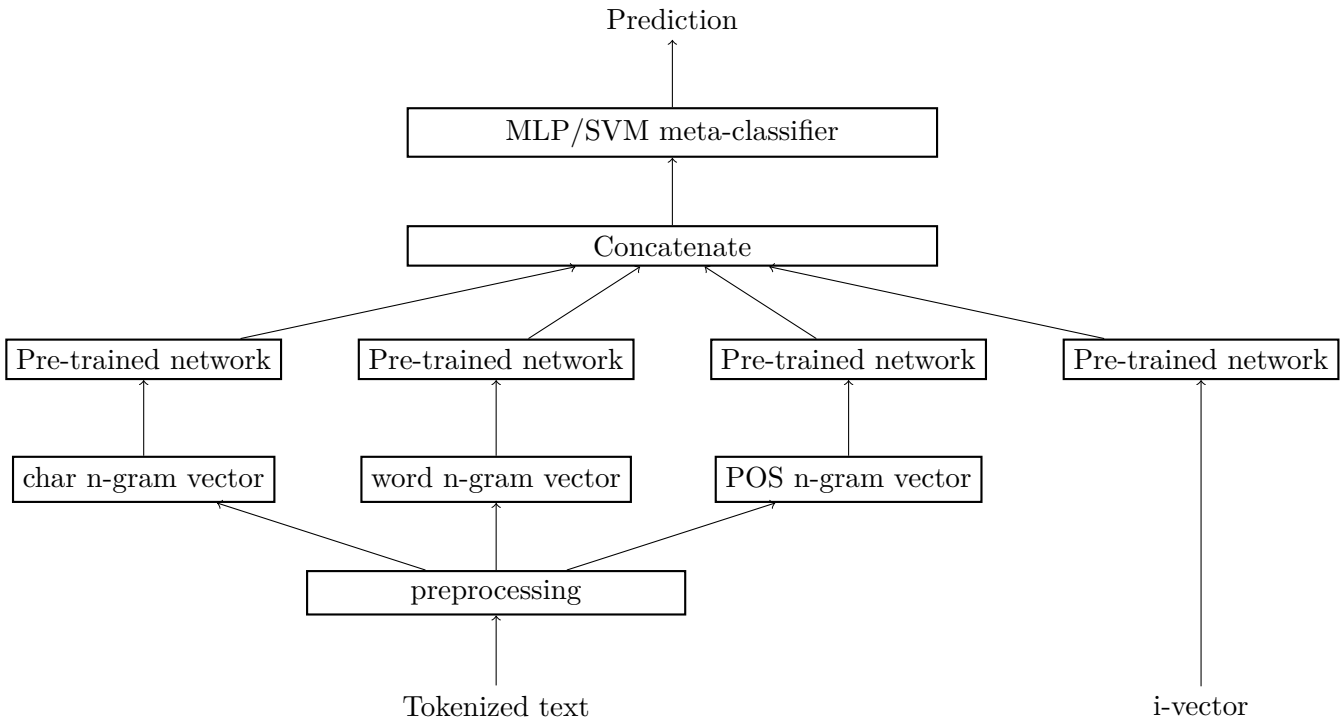

Figure 7: Simple Shallow Neural Network Ensemble Classifier.

Feature	Accuracy		Feature	Accuracy		
	1 Layer	2 Layers		1 Layer	2 Layers	2-Layer meta-classifier
Character	73.27%	77.06%	Character + Word	76.27%	78.91%	60.36%
Word	68.18%	78.73%	Character + POS	74.27%	78.36%	49.64%
POS	45.27%	58.09%	Word + POS	69.27%	73.64%	57.09%
i-vector	72.09%	75.55%	All n-grams	77.00%	80.64%	60.81%
			All + i-vector	79.18%	84.18%	72.09%
			Char + i-vector	77.36%	81.82%	69.09%

Table 7: Initial Results using hold-out validation with the development set from the 2017 Shared Task as the test set.

5 Experiments

A goal of this thesis is to find out whether a multilayer perceptron classifier setup can work well for NLI. As such the first step is to find out which multilayer perceptron classifier setup works best. The specific preprocessing method was less important for this test, since this is a classifier test. Therefore, since linear normalization was the simplest to implement, this is the one that is used. The results will also serve as a baseline. For all tests, the feature vector size for the classifiers are 30,000, 30,000, and 10,000 for character, word, and POS n-grams respectively. The n-gram sizes are character (3-6)-grams, word (1-4)-grams, and POS (1-3)-grams. Also, for the two layer perceptron classifiers, the size of the hidden layer is kept to 500, except for the meta-classifier which uses a hidden layer size of 20.

After a good classifier setup is found, the experiments move on to testing different layer sizes, feature vector sizes, and n-gram lengths, to find their impact on the accuracy of the system, and to find a good setup for the final system. As part of these experiments, an experiment is also run to see whether an individual long character n-gram feature vector should be used due to a possible exclusion of long character n-grams in the list of most common n-grams.

A set of experiments are then run using tf-idf weighting and χ^2 feature selection and comparing the results to the ones we found initially. These are expected to be better for the standard testing, but comparisons are also run where the test set instead consists of texts written to specific writing prompts and the rest being the training set. This will give a comparison of how the two methods perform on texts about unseen topics. For this test, the system again reverts to using layer sizes of 30,000, 30,000, and 10,000 for char, word, and POS, and uses the best setup found in the initial experiments.

Finally, a mixed preprocessing method is proposed and tested. This preprocessing is used by both the single multilayer perceptron classifier type as well as the ensemble, and the experiments focus on how the new preprocessing method performs when faced with unseen topics. This is done in the same way as the preceding experiments, by using texts written as answers to a single prompt as the test set. These results are compared

to tf-idf weighting and χ^2 feature selection as used in the previous experiment.

5.1 Choosing a classifier

To begin with, a good classifier has to be selected. As mentioned in section 4, the tested system consists of a set of fully connected perceptron classifiers. These classifiers were tested on individual features, as well as combinations of features. Since the specifics of the preprocessing is not important for this initial experiment, and due to the ease of implementing the k-most common and linear weighting, that is what is used for these experiments.

5.1.1 Initial results

The initial results can be seen in table 7, and seem quite promising considering how simple this preprocessing method is. As we can see, an extra layer helps for all classifiers (a similar conclusion was made by Li and Zou (2017)). For word and POS classifiers especially, the accuracy goes up significantly (POS performs more than 25% better with two layers, and word more than 15%). The accuracy is promising for multiple feature vectors classified by a single MLP classifier, even with such a simplistic preprocessing method. 80.64% is slightly below the rank 3 teams in the 2017 Shared Task for essay-only, and 84.18% is also comparable to rank 3 teams in the fusion track. Note that for all two layer classifiers, the classifier had completely fit the training data at the end of its training. This leads to the conclusion that using more than two layers is unnecessary, at least using this preprocessing method, and would probably negatively impact results due to overfitting.

However, something is wrong with the ensemble. The results when using a two layer meta-classifier on the single layer individual feature classifiers is consistently *worse* than the best individual classifier included. At this point, there were two hypotheses for why this happened. One was that the individual classifiers were interfering with each other due to making too many similar predictions, while the other was that the meta-classifier got stuck in a local maximum, and an alternative meta-classifier might improve results.

5.1.2 Classifier Homogeneity

To check whether the individual classifiers are interfering with each other, it was necessary to find out to which extent the n-gram classifiers' predictions overlapped with each other. A Venn diagram of the results can be seen in figure 8. Two conclusions can be taken from looking at these figures. First, classifier homogeneity probably does have an effect, since there is little doubt that there is a lot of overlap, especially compared to the relative uniqueness of the i-vector classifier. Second, there is still new information from all the individual classifiers. Some overlap is obviously to be expected, since all the classifiers use text data and are trained on the same data set. This means that it should be possible for a classifier to take advantage of this information, but the MLP classifier is clearly not the right one.

(a) (b)

Figure 8: Venn diagrams showing (a) the overlap for correct predictions between the three single layer n-gram classifiers, and (b) for the correct predictions of character, word, and i-vector classifiers.

Individual classifiers	Single layer	Two layers
Character + Word	72.55%	78.27%
Character + POS	70.82%	75.82%
Word + POS	69.64%	76.36%
All n-grams	73.00%	78.18%
All + i-vector	84.36%	85.82%
Char + i-vector	83.64%	84.64%

Table 8: Accuracy using a linear SVM meta-classifier instead of the two layer fully connected perceptron meta-classifier. Apart from the meta-classifier the system in the first column is identical to the system used by the meta-classifier in table 7. The results in the second column were achieved by training the meta-classifier on the output from two layer individual classifiers.

5.1.3 Alternative meta-classifier

A single MLP meta-classifier trained on all the individual classifiers reached an accuracy of 84.45%. Repeated attempts to replicate this result failed, which leads to the conclusion that the meta-classifier easily gets stalled in a local optimum. To confirm this, an experiment was run using a linear SVM classifier as the meta-classifier instead. The system was otherwise identical, except a second system using two layer individual classifiers was also tested, since the two layer classifiers consistently gave better results. The hope was that an SVM classifier would have more stable results, and therefore give a better idea of the performance of the input preparation. The complete results can be seen in table 8, and, as expected, the ensemble performs much better, probably since it is not stuck in local optima.

The best neural classifier achieved a result of 84.45%, which repeated attempts were

Feature	Accuracy
Character (3-9), vector size 30000	77.18%
Character (3-6), vector size 45000	77.45%
Character (3-9), vector size 45000	77.18%
Character (7-9), vector size 15000	70.81%
Character-only Ensemble, (3-6) and (7-9)	76.09%
Word (1-5), vector size 30000	78.36%
Word (1-4), vector size 45000	79.55%
Word (1-5), vector size 45000	79.36%
POS (1-4), vector size 10000	58.27%
POS (1-3), vector size 15000	55.36%
POS (1-4), vector size 15000	54.55%

Table 9: Accuracy for individual classifiers and ensemble using longer n-gram sizes.

unable to replicate. Due to this, it seems reasonable to assume that the SVM meta-classifier result of 84.36% is probably close to the optimal result using this input preparation. Also expected, but of note, is that using the two layer individual classifiers gives an across the board increase in ensemble accuracy compared to using single layer classifiers. This is especially noticeable for the *word + POS* ensemble, which can be assumed to be because of the significant increase in accuracy for the two layer n-gram classifiers for both word and POS inputs when compared to their single layer counterparts.

However, it is important to note that the meta-classifier is still only better than using concatenated input two layer MLP classifiers when including i-vectors, and even then, only slightly. The reason for this improvement when using the i-vector is probably due to the increased importance of the i-vector input in an ensemble system compared to concatenation (11 out of 44 compared to 800 out of 70,800) coupled with the high amount of new information from the i-vector classifier, as seen in figure 8b. Unless stated otherwise, all future experiments use the two layer individual classifiers with an SVM meta-classifier.

5.2 Refining the classifier setup

The next set of experiments focused on looking at the effect of larger n-grams and larger feature vector sizes, as well as looking at different hidden layer sizes for the character n-gram classifier. This was in order to find a good setup for the system before moving on to the most important set of tests.

5.2.1 Larger n-gram and feature vector sizes

The first part of these experiments focused on the effect of using larger n-grams for the individual classifiers. This is done in two main ways. First, the length of the n-grams are simply increased (from 3-6 to 3-9 for character, 1-4 to 1-5 for word, and 1-3 to 1-4 for POS), while keeping the length of the feature vectors the same. For the second setup

Feature	Vector Size	Accuracy
Character	100000	79.36%
Word	100000	78.36%
POS	20000	53.45%

Table 10: Results of a more extensive size increase of the feature vectors.

feature vector sizes are increased (to 45,000 for character and word, and 15,000 for POS). In order to get a set of baseline values for these vector lengths to compare to, tests were first run using the original n-gram sizes, and then repeated using the larger n-gram sizes. For character n-grams an experiment was also run using an extra classifier for n-grams of length 7-9.

As we can see from the results in table 9, none of these changes had much of an effect. This might not be very surprising with regards to the longer n-grams, since the way the items were chosen is so simplistic, but it is surprising that even the longer feature vector size seems to have a fairly negligible effect on accuracy. For the POS vector, the larger vector size even seems to have had a negative impact. Another surprising finding was that all the 15,000 most common n-grams of length 7-9 were also included in the 45,000 most common n-grams of length 3-9, leading to the conclusion that there is no need for an individual classifier for long character n-grams.

5.2.2 Larger vector sizes

Since the larger vector size seemed to have a negligible effect for this preprocessing method, another set of experiments was run to see whether a large increase in feature vector size would have any effect. These results were found with the n-gram sizes reverted to their initial values. As we can see from table 10, this seems to increase the accuracy for the character n-gram classifier somewhat, while the result for the word n-gram classifier was negligible, and again, for the POS classifier, the results seem to have become worse with the increased size.

5.2.3 Hidden layer sizes

Using different sizes of hidden layers is another variable that has an impact on the accuracy of the model. To explore this impact and choose a good hidden layer size, a few experiments were run using two different feature vector sizes. These experiments used, like for the vector size experiments, character n-gram classifiers. As we can see from the results in table 11, using very large layer sizes only gave reduced results. These poor results are actually not due to overfitting, since the results on the training set are only slightly better. The conclusion of these results is that there is no reason to deviate from the chosen hidden layer size of 500 for this system.

Feature vector size	Layer size	Accuracy
30000	500	**77.06%**
30000	250	76.54%
	1000	69.91%
	5000	56.55%
	10000	30.45%
100000	500	79.36%
100000	50	77.36%
	250	**79.91%**
	1000	79.63%
	5000	46.27%

Table 11: Character n-gram two layer classifier with different feature vector and layer sizes. The results using the hidden layer size of 500 is the baseline from preceding experiments.

		Testing set					
Preparation	Feature	dev	P0	P1	P2	P3	P4
tf-idf/χ^2	Character	79.45%	63.15%	56.35%	60.96%	61.27%	63.95%
	Word	77.73%	65.07%	63.54%	67.20%	58.95%	58.94%
	POS	54.36%	43.67%	45.42%	49.02%	52.14%	54.71%
	Ensemble	80.18%	66.33%	61.56%	71.72%	71.35%	71.35%
	+ i-vector	87.45%	N/A	N/A	N/A	N/A	N/A
linear/common	Character	77.05%	62.29%	55.00%	60.31%	61.21%	66.61%
	Word	78.73%	71.17%	63.23%	70.41%	72.66%	75.24%
	POS	45.27%	48.11%	45.62%	45.16%	54.80%	51.87%
	Ensemble	78.18%	67.13%	60.00%	67.08%	69.28%	70.47%
	+ i-vector	85.82%	N/A	N/A	N/A	N/A	N/A

Table 12: A comparison of results using two different input preparation methods on the development set, as well as using prompts as the test set instead. The prompts here are labeled using the labels from the 2017 Shared Task, a conversion to the prompts from the original TOEFL11 can be found in section 3.2.

Feature	tf-idf	linear
Character	78.36%	74.91%
Word	76.45%	77.64%
POS	58.45%	56.82%
i-vector	74.91%	
All	79.45%	81.09%
Ensemble	83.45%	84.09%

Table 13: Accuracy scores for SVM-only classifiers.

5.3 Input preparation

Another goal for the thesis was to explore whether an alternative input preparation method could improve results on texts about unseen topics. Some experiments therefore needed to be run to compare the simplistic linear input preprocessing to a more traditional input preparation method, and comparing the results on both seen and unseen prompts. As mentioned, the traditional preprocessing method chosen for this system is tf-idf weighting and choosing the k-best items using a χ^2 metric. Both are implemented in scikit-learn. Look at the first colum of table 12. As expected, for the baseline results, using the development set as the test set, the more traditional input preprocessing method is better on seen topics for all classifiers, except the word classifier, which might simply be an outlier. Surprisingly, the results were not as much of an improvement as was expected.

5.3.1 Alternative ensemble

Since the baseline results for the tfidf/χ^2 input preprocessing were worse than expected, a test was run to see if this was due to a weakness in using MLP classifiers with tfidf-weighted input. Using linear SVM classifiers for all parts of the input using both input preparation methods, we can see (table 13) that the results are slightly worse than using the MLP classifiers. This leads to the conclusion that the results are probably due to the fairly small feature vector sizes and the overall simplicity of the system.

5.3.2 Dropping prompts from the training data

As we can see from the rest of the results in table 12, there is a decrease in accuracy for all individual classifiers no matter the input preparation method. However, for the word classifier there is in fact less of a drop when using the most common items instead of the traditional preprocessing method. While this may seem surprising, topic words are an obvious source of this discrepancy, and since there is a huge amount word n-grams compared to character and POS n-grams, the 30,000 most common items are very unlikely to contain many items that are only useful for a specific topic. However, when chosen for most impact, there is a good chance that many of the included words will only be useful for specific prompts. This difference probably means that the χ^2

selection has a greater ratio of prompt specific words in its set of 30,000, and the most common words are an equal or better choice depending on the prompt. However, the weakness of the most common items shows when it comes to the ensemble classifier, where the accuracy is consistently worse. This is probably because of less homogeneity in the individual classifiers when using tf-idf weighting and χ^2 feature selection, which leads to a better ensemble result.

5.4 Final system design

The final step was to combine the information gained from the preceding experiments. Since tf-idf/χ^2 feature preprocessing seems to usually be better for character and POS classifiers, while choosing the most common items seems to usually be better (in many cases a lot better) for the word classifier, the idea for the final system was to combine this into one system. The expectation was that while this system might be worse on prompts for which the system has been trained, it will perform better than using pure tf-idf/χ^2 feature preprocessing on prompts for which it has not been trained. Since changing the vector sizes and n-gram sizes had a minimal effect on overall accuracy, all of these were kept at their initial values to give a better idea of the value of the mixed preprocessing method. The test was run using both the ensemble method and a single MLP trained on the concatenated feature vectors. The results when using all items from a single prompt as the test set can be seen in tables 14, 15, and 16, including precision, recall, and F1 scores for individual languages as well as the whole test set. In addition to looking at results on unseen prompts, a 10-fold cross-validation test was run using the mixed preprocessing method. The results of this cross-validation test can be seen in table 17, and a combined confusion matrix for the cross-validation can be seen in figure 9. Prompts P6 and P7 were not tested, since P0 through P5 were the prompts that were expected to give the most trouble. These six include the prompt that creates the largest test set and the prompt that has the worst balance in terms of L1s. Other large prompts and another imbalanced prompt were also included to confirm any issues that may have been due to those considerations. The results from prompts P6 and P7 would therefore give less new information about the performance of this method. The results of the final experiments are discussed in the next section.

	Essay-only											
	dev			P0			P1			P2		
L1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
ARA	0.79	0.80	0.80	0.85	0.76	0.80	0.74	0.78	0.76	0.66	0.81	0.73
CHI	0.80	0.82	0.81	0.78	0.84	0.81	0.88	0.84	0.86	0.75	0.72	0.73
FRE	0.87	0.84	0.85	0.80	0.76	0.78	0.81	0.81	0.81	0.92	0.61	0.73
GER	0.82	0.93	0.87	0.80	0.94	0.87	0.87	0.93	0.90	0.85	0.86	0.85
HIN	0.68	0.75	0.71	0.72	0.72	0.72	0.74	0.80	0.77	0.61	0.76	0.67
ITA	0.87	0.82	0.85	0.89	0.79	0.84	0.85	0.83	0.84	0.80	0.79	0.80
JPN	0.80	0.82	0.81	0.71	0.70	0.71	0.80	0.73	0.77	0.84	0.40	0.54
KOR	0.82	0.75	0.78	0.83	0.70	0.76	0.72	0.70	0.71	0.71	0.73	0.72
SPA	0.80	0.66	0.72	0.66	0.78	0.72	0.78	0.76	0.77	0.76	0.75	0.75
TEL	0.76	0.75	0.75	0.80	0.80	0.80	0.83	0.78	0.80	0.79	0.73	0.76
TUR	0.78	0.81	0.79	0.84	0.88	0.86	0.74	0.77	0.76	0.57	0.81	0.67
Avg	0.80	0.80	0.79	0.79	0.79	0.79	0.80	0.80	0.80	0.75	0.73	0.72
Accuracy	79.55%			79.01%			79.75%			72.61%		

Table 14: Precision, Recall, F1, and Accuracy for the ensemble on essay-only features when using the mixed preprocessing method.

	Essay-only								
	P3			P4			P5		
L1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
ARA	0.83	0.79	0.81	0.76	0.70	0.73	0.81	0.68	0.74
CHI	0.88	0.90	0.89	0.99	0.51	0.67	0.72	0.79	0.75
FRE	0.82	0.85	0.84	0.86	0.84	0.85	0.76	0.75	0.76
GER	0.88	0.93	0.90	0.59	0.97	0.73	0.56	0.97	0.71
HIN	0.74	0.82	0.78	0.67	0.80	0.73	0.71	0.45	0.55
ITA	0.86	0.89	0.88	0.85	0.78	0.81	0.86	0.84	0.85
JPN	0.84	0.79	0.81	0.78	0.61	0.69	0.80	0.78	0.79
KOR	0.82	0.80	0.81	0.59	0.91	0.72	0.94	0.34	0.49
SPA	0.85	0.79	0.82	0.71	0.58	0.64	0.59	0.61	0.60
TEL	0.86	0.75	0.81	0.80	0.59	0.68	0.72	0.73	0.73
TUR	0.82	0.86	0.84	0.88	0.79	0.83	0.64	0.84	0.73
Avg	0.84	0.83	0.83	0.77	0.74	0.74	0.74	0.71	0.70
Accuracy	83.39%			74.09%			71.27%		

Table 15: Precision, Recall, F1, and Accuracy for the ensemble on essay-only features when using the mixed preprocessing method continued.

	Test set						
Classifier	dev	P0	P1	P2	P3	P4	P5
MLP Character + Word + POS	82.73%	81.57%	80.83%	70.59%	78.10%	75.12%	73.48%
+ i-vector	80.82%	N/A	N/A	N/A	N/A	N/A	N/A
Ensemble tf-idf/χ^2 essay-only	80.18%	66.33%	61.56%	71.72%	71.35%	71.35%	74.51%
+ i-vector	87.45%	N/A	N/A	N/A	N/A	N/A	N/A

Table 16: Accuracy of the alternative classifiers: MLP with the mixed input preprocessing, and Ensemble with the pure tf-idf/χ^2 preprocessing.

Classifier	Accuracy
Ensemble (essay-only)	80.41%
Ensemble (fusion)	87.25%
MLP (essay-only)	82.56%
MLP (fusion)	75.69%

Table 17: 10-fold cross-validation accuracy of the two different systems when using the mixed preprocessing method.

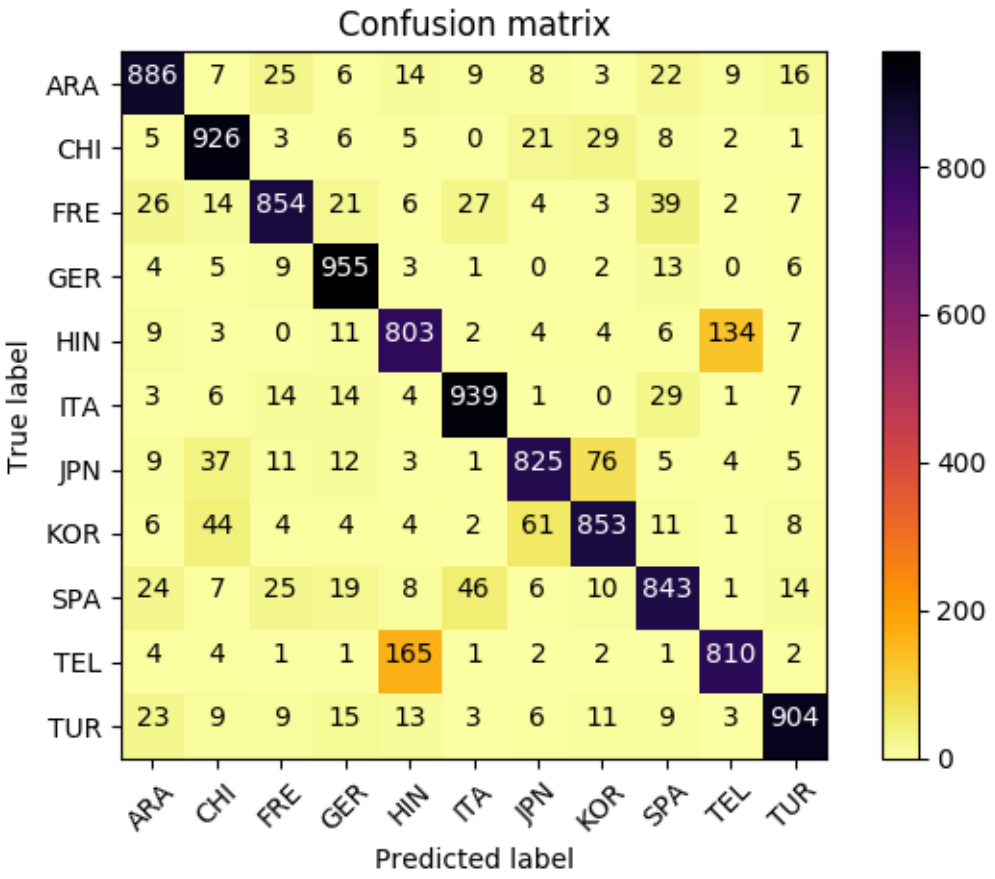

Figure 9: Combined confustion matrix for the fusion system when tested using 10-fold cross-validation.

6 Discussion

As we see in tables 14, 15, and 16, the mixed input preparation method usually performs better than tf-idf/χ^2 preprocessing on unseen data. For P3, the system even performs better than it does on the dev set. The expected lowered performance on the dev set is not particularly bad. P3 is also the only prompt for which the MLP classifier on the concatenated feature vector performs worse than the ensemble. However, for P5 the input preparation performs worse than tf-idf/χ^2. There was no time to look into what was different with this prompt, but intuitively, this prompt is probably less prone to topic word bias than the other prompts. It could also have been due to a large amount of essays, but other prompts with similar or larger sizes, P3 and P4, have not had this large drop in accuracy, which rules set size out as the main reason.

Looking at the precision and recall scores for the different prompts, there is a great deal of variance, and it is not immediately obvious why. For example, the recall score of Hindi for P5 is 0.45, meaning that Hindi is guessed less than 50% of the time it appears. If there was an imbalance in the share of Hindi essays in the test set compared to the training set, one could easily understand such an issue, but P5 is a fairly balanced prompt, and the scores for Italian, which has a greater degree of imbalance, show no such issues. When recall is low on one language, that necessarily drags down precision for other classes, and for Hindi, the obvious target would be Telugu, which is usually its most mutually confused native language (figure 9). However, there does not seem to be a very large drop in precision for Telugu, while German, which has previously seen very little confusion with Hindi has a significant drop in precision, while keeping recall high, meaning it is guessed a lot more often than it should. Looking at the confusion matrix for this prompt (figure 10) and comparing it to the combined confusion matrix (figure 9), it is clear that German is guessed across the board more often than normal.

My first hypothesis was that this might be due to higher proficiency among writers of this prompt, and that the German linguistic similarity to English makes it the most similar to other high proficiency texts. Comparing the confusion matrix from the mixed preprocessing to the confusion matrix when using the traditional method, seen in figure 11, makes it clear that it is not quite as simple as that. However, the same L1s (Hindi, Korean, and Spanish) have an abnormally low recall, so there is clearly some quirk regarding this prompt and those languages.

Looking at other confusion matrices from the other prompts (found in appendix A), we can see that some prompts seems to have different languages for which the recall is low, and Japanese and Spanish have this problem for two prompts. There are clearly some peculiarities regarding each prompt and the languages, and it is not readily apparent from essay counts why the confusion for these languages should be this way for the prompts. There was unfortunately not enough time to explore this further for this thesis, but a thorough exploration of the prompts with regards to language needs to be undertaken.

Table 17 shows the results of the final system when tested using 10-fold cross validation, and as we can see, the system has good results. The classifiers are well within

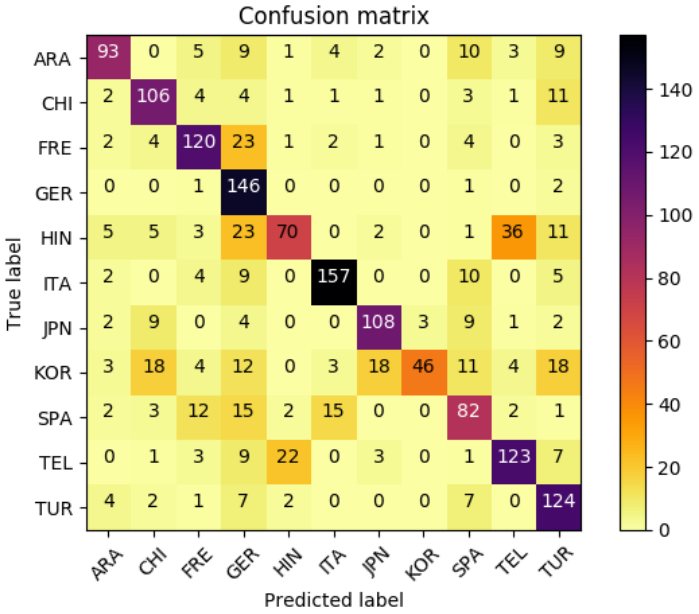

Figure 10: Confusion matrix for texts written to prompt P5 when trained on the rest of the essays using the mixed preprocessing method.

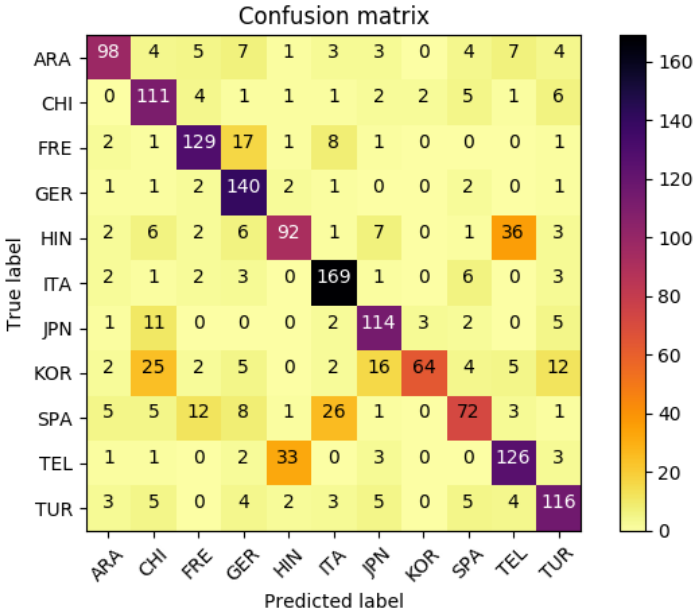

Figure 11: Confusion matrix for texts written to prompt P5 when trained on the rest of the essays using tf-idf weighting and χ^2 feature selection on all features.

rank 3 systems from the 2017 Shared Task for both essay-only and fusion tracks, which shows that a simple system design and preprocessing can give acceptable results in NLI.

Looking at the combined confusion matrix from the 10-fold cross-validation, seen in figure 9, there are no surprising results when looking at other systems developed for NLI. The confusion often seems to be more related to geographical vicinity rather than linguistic similarity. For example, the two Indian languages Hindi and Telugu are the most mutually confused by the system, even though these two languages come from two different language families (Indo-European and Dravidian, respectively). At the same time, some languages that are geographically close *and* linguistically close, are confused less often.

What could be the reasons for these results? German is the least confused language. My hypothesis for this is that this is probably due to the linguistic relatedness of German to English, which might make it possible for German native speakers to directly use some of the German language structures and word choices in English while keeping the sentence at a fairly competent level. A language that is farther away from English linguistically would have more trouble doing this, which might mean that mistakes that are more general become more common. Since the average essay lengths are similar between the languages (Blanchard et al. 2013), this is unlikely to be a cause of this discrepancy. A reason the two Indian languages have such a high degree of mutual confusion, could also be because English is such a common second language in India as a whole, and that it might simply be because of the way in which English is taught to Indian students.

6.1 Simple classifier

Considering the simplicity of the classifiers and the fairly small feature vectors, both the ensemble and the MLP classifier worked very well. Both of them have an accuracy that would place them well within rank 3 teams in the 2017 Shared Task. However, there is a long way up to state of the art results for this model. One of the main reasons for this is probably the very small feature vector sizes. As we can see from section 5.2.2 and table 10, increased vector sizes do have an effect, but at these still fairly small sizes the improvement was minor. As such, the next step would probably be to look at a significant increase in vector sizes (10-20 times) to see what effect that would have on the result.

One thing to note is that, just as in the earlier experiments, the simplest model (a single MLP classifier) gives better results for essay only-data, while getting less of a boost when i-vectors are also included, which leads to the ensemble performing best for the fusion track. It might be possible to combine these strengths into an even better classifier.

6.2 Preprocessing

Using the ensemble, the traditional tf-idf/χ^2 preprocessing method faces a drop in accuracy of between 7% and 30%. As we saw, using frequency based feature selection and

linear weighting alone did not achieve better results. However, the hybrid version used in the final system, which only performs 1% worse on previously seen prompts, only faces a drop of up to 12%, and even has no drop in accuracy for certain prompts, a significant improvement. The drop in accuracy is not always better on every prompt, however, as we saw with the results on prompt P5.

While the performance of frequency based selection is good for word n-grams in particular, it gave less of an improvement to the other essay n-gram features (character and word). A look at a frequency based character n-gram showed that a lot of the features that were right next to each other in the vector were simply the same n-gram with a single space or other character added. This means that there is a lot of data in the feature vector that are not independent of each other. Ideally, to improve the performance on character n-grams, a method would need to be used that kept the strength of frequency based selection (fewer topic words) while also avoiding this waste of space in the feature vector.

7 Conclusions and Future Work

This thesis started with two main goals: to explore the use of MLP for NLI and to find an input preprocessing method that reduced drops in accuracy when faced with topics for which the NLI classifiers had not been trained. This section will draw some conclusions about how this thesis fulfilled those goals, and answer the three research questions that were introduced in the introduction.

7.1 Conclusion

This thesis explored the use of shallow MLP classifier systems for Native Language Identification, and proposed two straightforward architectures. The simplest is a two layer MLP classifier, which, of the two models, performs the best on essay-only features, and would be the recommended continuation of this approach, even more so because of its simplicity. If feature selection and preparation is changed significantly, for example by using several methods for the same feature, an ensemble might achieve better performance.

Two questions were posed relating to this goal in the introduction of this thesis. The first was *to what extent can MLP be used effectively for NLI?* Using only these small feature vector sizes, a single MLP classifier achieved results comparable to rank 3 systems in the 2017 Shared Task for essay-only data, and when combined in an ensemble, also achieved rank 3 results when adding i-vector speech data. As such, it is clear that MLP can be quite effective at this task, especially when the small feature vector sizes limit the long training times that might otherwise be a problem for this model. Looking at the results for SVM classifiers on the same data, seen in table 13, it is clear that the two layer perceptron classifiers consistently outperform SVMs on these feature vectors.

The second, smaller, question was *how many layers are necessary for a trained MLP to completely fit the training data?* The answer to this question was found quite quickly. Two layers were enough to fit the data, and as such, adding more layers would defeat the purpose, and only serve to make the model overfit.

The second part of this thesis is the attempt at improving the results on unseen prompts using simpler preprocessing methods. When using the most common terms and linear normalization of the feature vector, only the word classifier gained a consistent improvement on unseen prompts compared to more traditional tf-idf weighting and χ^2 feature selection. The other classifiers had drops that were equally bad for both input preparation methods, and lower homogeneity leads to better overall results using tf-idf/χ^2. When combining the simpler word preprocessing with traditional preprocessing for the other features, the system gave improved inputs for most prompts. However, for prompt P5, the traditional input methods performed better. There was not enough time to explore why this input gave such different results from the others, but, intuitively, prompt P5 could be less prone to topic words compared to the others. Exploring this would be the obvious next step.

The partial success of the combination method leads me to the conclusion that improvements on unseen prompts is very possible by using vectors that avoid topic words.

A negligible drop seems within reach, although I suspect some of the reasons for the extremely low drop in accuracy for the system in this thesis on some prompts might also be paired with the initial lower accuracy compared to the best systems from the 2017 Shared Task.

7.2 Future Work

There were several ideas that needed to be abandoned in order to keep focus on the important parts of this thesis or because there was no time. The most pressing one is to explore why the combined preprocessing method works so well on some prompts while much less on others. My initial thoughts are that P5 is somehow a more general prompt, or that it is answered by more proficient test takers, but without enough time to thoroughly compare the essays and no direct knowledge of the actual prompts or proficiency scores by prompt these remain hypotheses. This should be explored before continuing to develop this preprocessing method, in order to better understand its weaknesses.

The most important of the abandoned ideas was a more in depth alternative preprocessing method, which was touched upon in 6.2. Frequency based feature selection worked better than expected, but, for character n-grams, a close inspection shows that a lot of the chosen n-grams are actually just different versions of what is essentially the same n-gram. The reason for this is obviously that the n-gram that is one item longer or shorter than a given n-gram would in all likelihood appear in a frequency very similar to that n-gram. Thus, to keep the strength of choosing the most common items (creating a feature vector that generalizes better), while avoiding the problem that what is essentially the same feature can appear at several locations in the vector, it might be interesting to experiment with merging these n-grams into one entry while still keeping the frequency based feature selection.

The MLP classifier on the concatenated feature vectors gave consistently better accuracy than the ensemble for essay-only data, and this could possibly be taken advantage of for a better classifier. The initial idea, which there was unfortunately no time to work on, is to train two MLP classifiers, one for essay data and one for i-vectors, and then combine these classifers using an SVM meta-classifier. This could potentially give improved accuracy compared to the current systems while still keeping the feature vector sizes small.

References

Abad, Alberto, Eugénio Ribeiro, Fábio Kepler, Ramon Astudillo, and Isabel Trancoso. "Exploiting phone log-likelihood ratio features for the detection of the native language of non-native English speakers". In: San Francisco, CA, USA, 2016, pp. 2413–2417.

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

Bird, Steven, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

Bjerva, Johannes, Gintare Grigonyte, Robert Östling, and Barbara Plank. "Neural networks and spelling features for native language identification". In: *Proceedings of the Twelfth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Copenhagen, Denmark, 2017, pp. 235–239.

Blanchard, Daniel, Joel Tetreault, Derrick Higgins, Aoife Cahill, and Martin Chodorow. *TOEFL11: A corpus of non-native English*. Tech. rep. Educational Testing Service, 2013.

Brooke, Julian and Graeme Hirst. "Native language detection with 'cheap' learner corpora". In: *Twenty Years of Learner Corpus Research. Looking Back, Moving Ahead: Proceedings of the First Learner Corpus Research Conference (LCR 2011)*. Vol. 1. Presses Universitaires de Louvain. Louvain, France, 2013, pp. 37–47.

Cimino, Andrea and Felice Dell'Orletta. "Stacked Sentence-Document Classifier Approach for Improving Native Language Identification". In: *Proceedings of the Twelfth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Copenhagen, Denmark, 2017, pp. 430–437.

Cortes, Corinna and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

Dehak, Najim, Patrick J Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. "Front-end factor analysis for speaker verification". In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.4 (2011), pp. 788–798.

Fan, Rong-En, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. "LIBLINEAR: A Library for Large Linear Classification". In: *Journal of Machine Learning Research* 9 (2008), pp. 1871–1874.

Gebre, Binyam Gebrekidan, Marcos Zampieri, Peter Wittenburg, and Tom Heskes. "Improving native language identification with tf-idf weighting". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Atlanta, GA, USA, 2013, pp. 216–223.

Goutte, Cyril and Serge Léger. "Exploring Optimal Voting in Native Language Identification". In: *Proceedings of the Twelfth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Copenhagen, Denmark, 2017, pp. 367–373.

Goutte, Cyril, Serge Léger, and Marine Carpuat. "Feature space selection and combination for native language identification". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Atlanta, GA, USA, 2013, pp. 96–100.

Granger, Sylviane, Estelle Dagneaux, Fanny Meunier, and Magali Paquot. *International corpus of learner English*. Louvain, France: Presses Universitaires de Louvain, 2002.

Hochreiter, Sepp and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

Ionescu, Radu Tudor and Marius Popescu. "Can string kernels pass the test of time in Native Language Identification?" In: (2017), pp. 224–234.

Jarvis, Scott, Yves Bestgen, and Steve Pepper. "Maximizing classification accuracy in native language identification". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Atlanta, GA, USA, 2013, pp. 111–118.

Koppel, Moshe, Jonathan Schler, and Kfir Zigdon. "Determining an author's native language by mining a text for errors". In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM. Chicago, IL, USA, 2005, pp. 624–628.

Kríž, Vincent, Martin Holub, and Pavel Pecina. "Feature Extraction for Native Language Identification Using Language Modeling". In: *Recent Advances in Natural Language Processing* (2015), pp. 298–306.

Kulmizev, Artur, Bo Blankers, Johannes Bjerva, Malvina Nissim, Gertjan van Noord, Barbara Plank, and Martijn Wieling. "The power of character n-grams in native language identification". In: *Proceedings of the Twelfth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Copenhagen, Denmark, 2017, pp. 382–389.

Lado, Robert. *Linguistics Across Cultures: Applied Linguistics for Language Teachers*. Ann Arbor, MI, USA: University of Michigan Press., 1957.

Li, Wen and Liang Zou. "Classifier Stacking for Native Language Identification". In: *Proceedings of the Twelfth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Copenhagen, Denmark, 2017, pp. 390–397.

Lynum, André. "Native language identification using large scale lexical features". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educa-*

tional Applications. Association for Computational Linguistics. Atlanta, GA, USA, 2013, pp. 266–269.

Malmasi, Shervin and Mark Dras. "Native Language Identification using Stacked Generalization". In: *arXiv preprint arXiv:1703.06541* (2017).

Malmasi, Shervin, Keelan Evanini, Aoife Cahill, Joel Tetreault, Robert Pugh, Christopher Hamill, Diane Napolitano, and Yao Qian. "A report on the 2017 native language identification shared task". In: *Proceedings of the Twelfth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Copenhagen, Denmark, 2017, pp. 62–75.

Malmasi, Shervin, Sze-Meng Jojo Wong, and Mark Dras. "NLI shared task 2013: MQ submission". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Atlanta, GA, USA, 2013, pp. 124–133.

Martinez, David, Oldřich Plchot, Lukáš Burget, Ondřej Glembek, and Pavel Matějka. "Language recognition in iVectors space". In: *Twelfth Annual Conference of the International Speech Communication Association*. Florence, Italy, 2011.

Nair, Vinod and Geoffrey E Hinton. "Rectified linear units improve restricted Boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. Haifa, Israel, 2010, pp. 807–814.

Oh, Yoo Rhee, Hyung-Bae Jeon, Hwa Jeon Song, Yun-Kyung Lee, Jeon-Gue Park, and Yun-Keun Lee. "A deep-learning based native-language classification by using a latent semantic analysis for the NLI Shared Task 2017". In: *Proceedings of the Twelfth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Copenhagen, Denmark, 2017, pp. 413–422.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *Proceedings of the 30th International Conference on Machine Learning*. Atlanta, GA, USA, 2013, pp. 1310–1318.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

Popescu, Marius and Radu Tudor Ionescu. "The Story of the Characters, the DNA and the Native Language". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Atlanta, GA, USA, 2013, pp. 270–278.

Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications. Association for Computational Linguistics. Atlanta, GA, USA, 2013.

Proceedings of the Twelfth Workshop on Innovative Use of NLP for Building Educational Applications. Association for Computational Linguistics. Copenhagen, Denmark, 2017.

Rosenblatt, Frank. "The perceptron: A probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), pp. 386–408.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–538.

Schuller, Björn, Stefan Steidl, Anton Batliner, Julia Hirschberg, Judee K Burgoon, Alice Baird, Aaron Elkins, Yue Zhang, Eduardo Coutinho, and Keelan Evanini. "The INTERSPEECH 2016 computational paralinguistics challenge: Deception, sincerity & native language". In: *Proceedings of Interspeech*. San Francisco, CA, USA, 2016.

Senoussaoui, Mohammed, Patrick Cardinal, Najim Dehak, and Alessandro L Koerich. "Native language detection using the i-vector framework". In: San Francisco, CA, USA, 2016, pp. 2398–2402.

Srivastava, Nitish, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of machine learning research* 15.1 (2014), pp. 1929–1958.

Swanson, Ben and Eugene Charniak. "Native language detection with tree substitution grammars". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*. Association for Computational Linguistics. Jeju Island, South Korea, 2012, pp. 193–197.

Tetreault, Joel, Daniel Blanchard, and Aoife Cahill. "A report on the first native language identification shared task". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Atlanta, GA, USA, 2013, pp. 48–57.

Tetreault, Joel, Daniel Blanchard, Aoife Cahill, and Martin Chodorow. "Native Tongues, Lost and Found: Resources and Empirical Evaluations in Native Language Identification." In: *Proceedings of COLING*. The COLING 2012 Organizing Committee. Mumbai, India, 2012, pp. 2585–2602.

Toutanova, Kristina, Dan Klein, Christopher D Manning, and Yoram Singer. "Feature-rich part-of-speech tagging with a cyclic dependency network". In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics. Edmonton, Canada, 2003, pp. 173–180.

Tsur, Oren and Ari Rappoport. "Using classifier features for studying the effect of native language on the choice of written second language words". In: *Proceedings of the Workshop on Cognitive Aspects of Computational Language Acquisition*. Association for Computational Linguistics. Prague, Czechia, 2007, pp. 9–16.

Tsvetkov, Yulia, Naama Twitto, Nathan Schneider, Noam Ordan, Manaal Faruqui, Victor Chahuneau, Shuly Wintner, and Chris Dyer. "Identifying the L1 of non-native writers: the CMU-Haifa system". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. Association for Computational Linguistics. Atlanta, GA, USA, 2013, pp. 279–287.

Wong, Sze-Meng Jojo and Mark Dras. "Exploiting parse structures for native language identification". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. Edinburgh, Scotland, UK, 2011, pp. 1600–1610.

Zhang, Xiang, Junbo Zhao, and Yann LeCun. “Character-level convolutional networks for text classification”. In: *Advances in neural information processing systems 28*. Montréal, Canada, 2015, pp. 649–657.

Zhang, Ye and Byron Wallace. “A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification”. In: *arXiv preprint arXiv:1510.03820* (2015).

A Confusion matrices by prompt

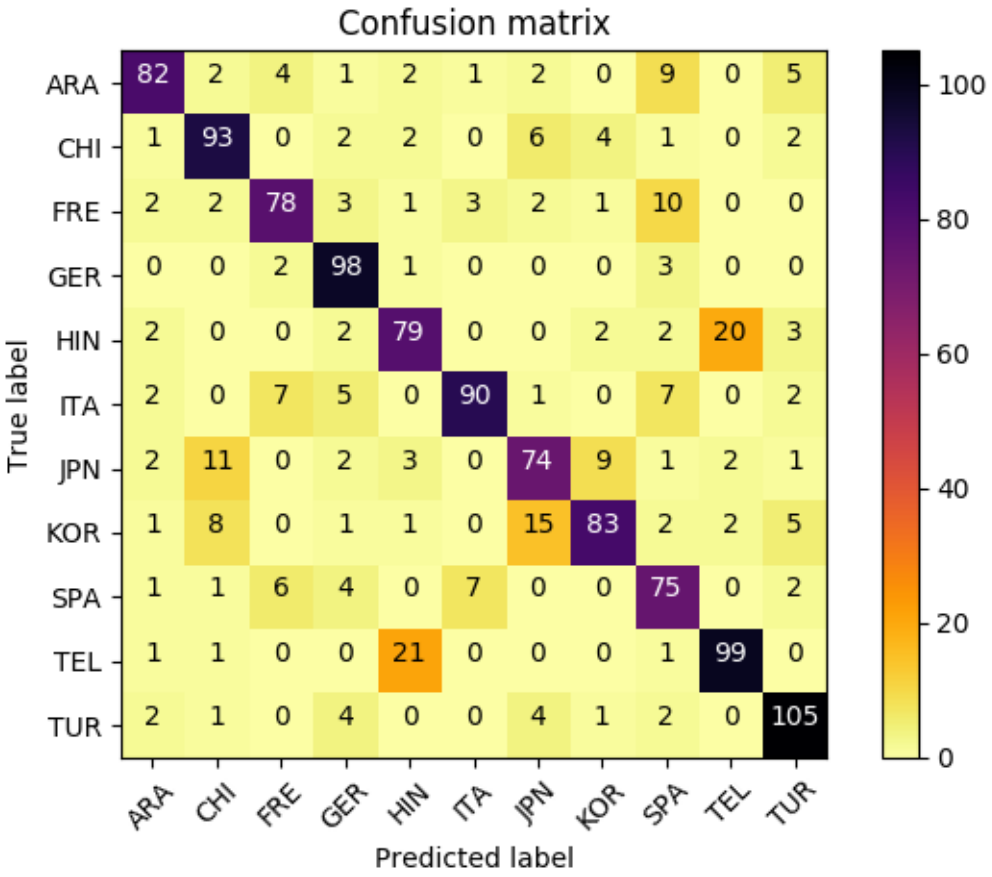

Figure 12: Confusion matrix for prompt P0

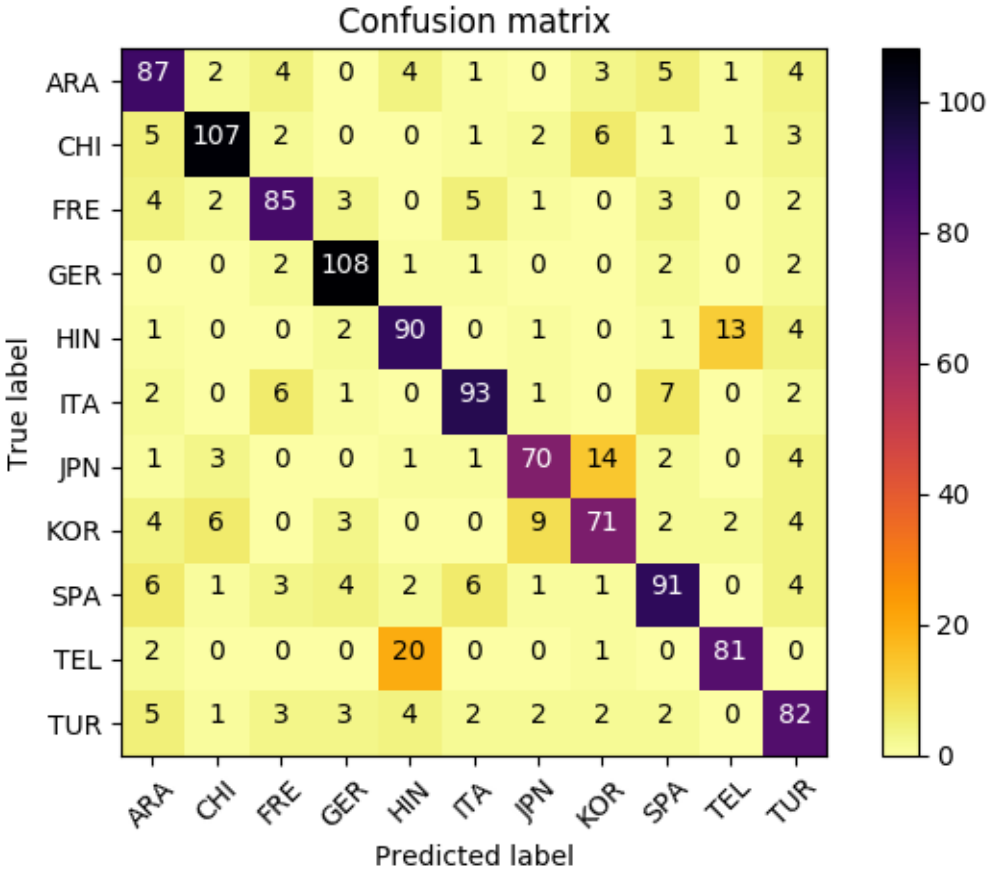

Figure 13: Confusion matrix for prompt P1

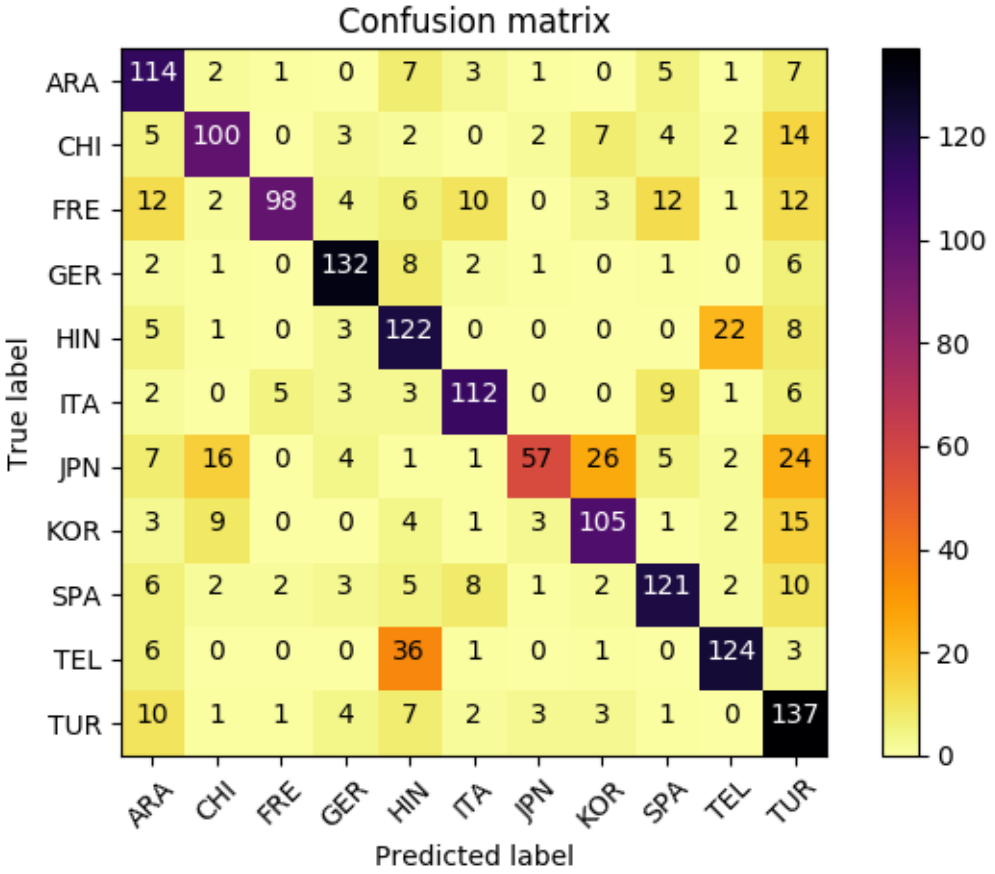

Figure 14: Confusion matrix for prompt P2

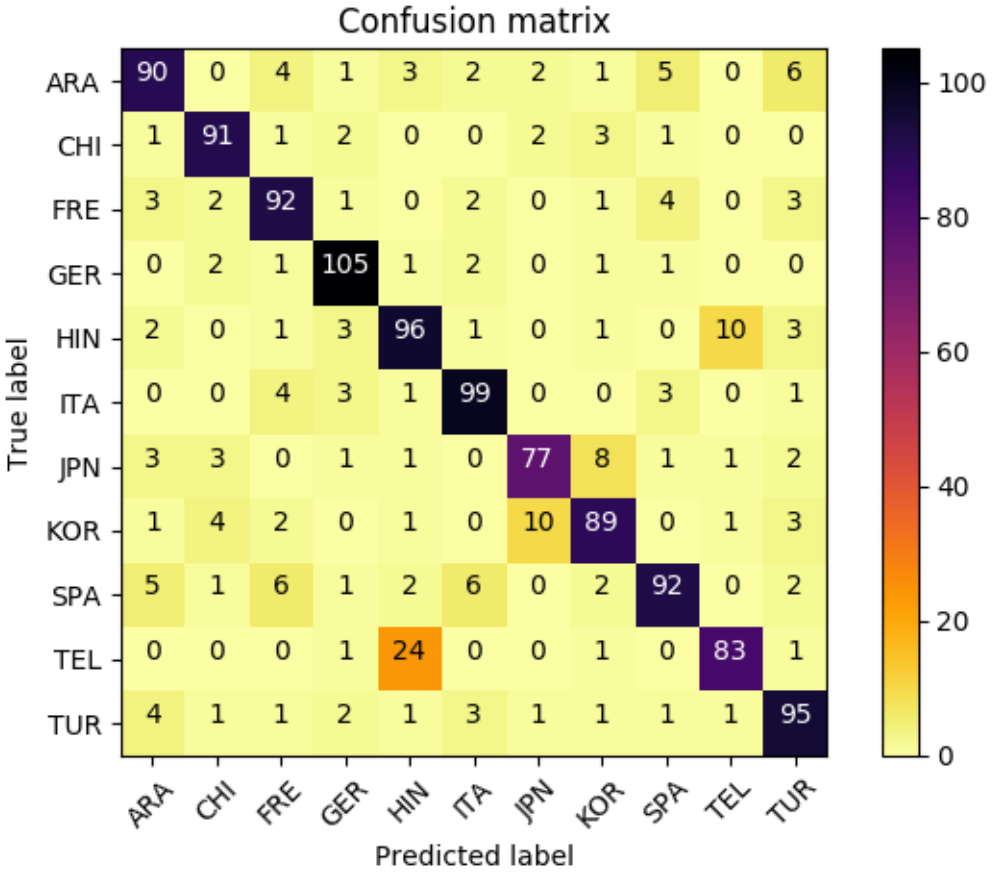

Figure 15: Confusion matrix for prompt P3

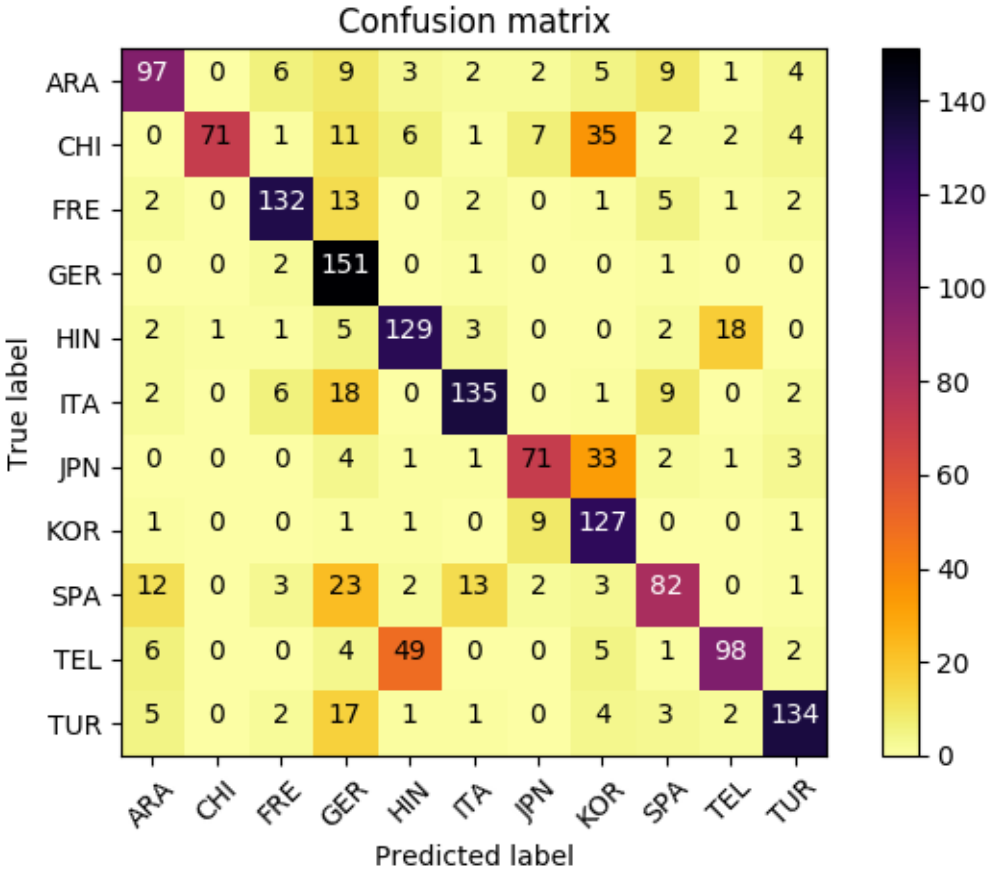

Figure 16: Confusion matrix for prompt P4

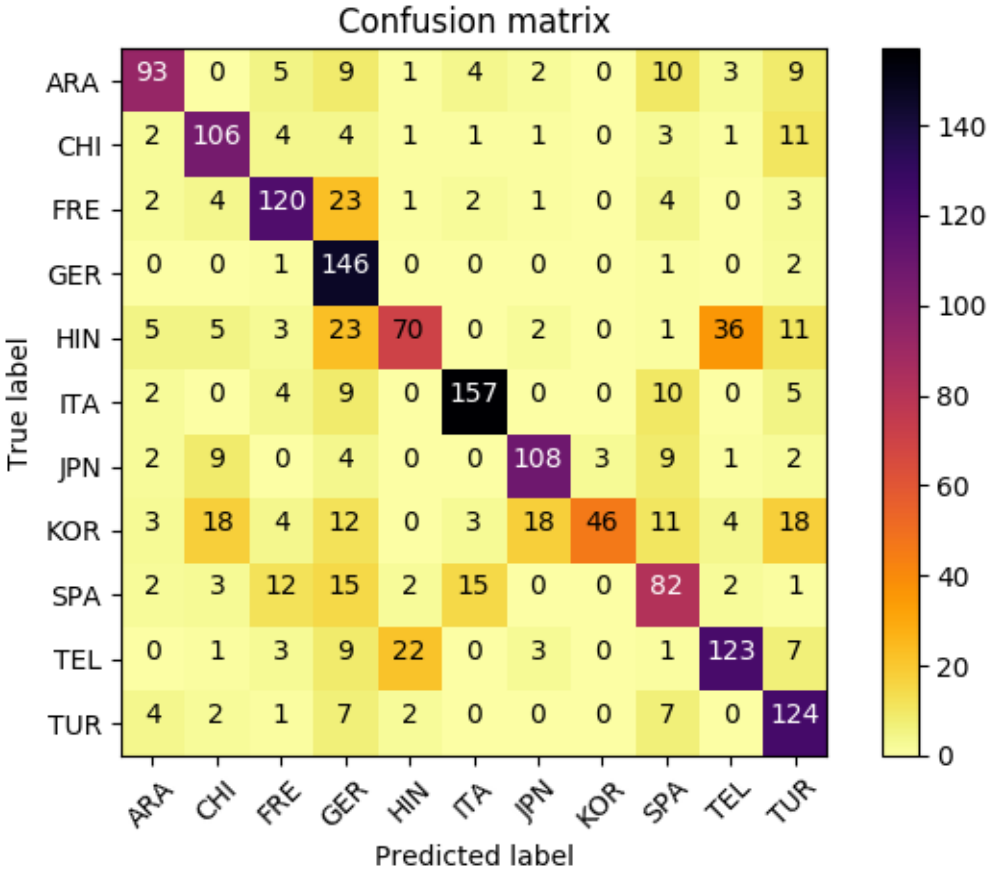

Figure 17: Confusion matrix for prompt P5