NTNU
Norwegian University of
Science and Technology

Gender prediction on Norwegian Twitter accounts

Håvard Kvamme

Master of Science in Physics and Mathematics
Submission date: December 2015
Supervisor: Håvard Rue, MATH
Co-supervisor: Dirk Hesse, Intelligent Communication AS

Norwegian University of Science and Technology
Department of Mathematical Sciences

Preface

This thesis completes my master's degree in industrial mathematics at Norwegian University of Science and Technology (NTNU).

I would like to thank my supervisors prof. Håvard Rue at NTNU and Dr. Dirk Hesse at Intelligent Communication AS, for their excellent understanding, guidance and support during the project. I also want to thank Intelligent Communication AS for allowing me to work with Dr. Hesse, and granting me access to both data and computational resources. Last, I want to thank Dr. Erlend Aune for introducing me to the Caffe deep learning framework.

Trondheim, 21.12.2015

Håvard Kvamme

ABSTRACT

In this thesis, methods for predicting the gender of Norwegian Twitter accounts were investigated. Through Twitter's public APIs, various account information is available. Tweets (text), personal descriptions, friends networks, and profile images were the main fields investigated. First separate classifiers were fitted to features from the different fields, and later the individual classifiers' posterior probability estimates were combined to achieve increased accuracy. The datasets were labeled though comparison of the accounts' names and names in the Norwegian population. Subsets of accounts with very gender specific names were used for training and testing. The highest balanced accuracy obtained was around 0.89. This, however, required access to the accounts' profile images (85% of the data). Without images, the accuracy dropped to around 0.85.

SAMMENDRAG

Denne oppgaven omhandler metoder for prediksjon av kjønn på norske Twitterkontoer. Diverse informasjon er tilgjengelig gjennom Twitters API. Hovedvekten av analysen ble lagt på tweets (tekst), personlig beskrivelser, vennenettverk og profilbilder. Disse ble først undersøkt individuelt, for så å bli kombinert gjennom deres sannsynlighetsestimater i et forsøk på å minke feilraten. Datasettene ble merket (gutt/jente) ved å sammenligne brukernes navn med den norske befolkningen. Undergrupper av brukere med navn som sterkt antyder et av kjønnene, ble brukt til trening og testing. Den høyeste balanserte treffsikkerheten oppnådd var på 0.89, men dette krevde tilgang til brukernes profilbilder (85% av dataene). Uten bilder sank treffsikkerheten til 0.85.

Contents

Chapter 1

Introduction

Online social media networks have become a popular way for people to connect to each other. This can be in the form of network building, communication, or broadcasting content. The increasing popularity of the networks gives rise to extraordinary opportunities for studying the human society at scale. However, due to concerns for user privacy, many service providers keep much of the user data private. Twitter represents an exception.

Twitter is an online social networking service that enables users to post and read 140 character messages. These messages are commonly referred to as *tweets* [1]. According to Twitter's own web pages [2], the world wide network currently consists of 320 million active users monthly, and one billion unique visits monthly to sights with embedded tweets. In Norway there are currently a bit more than one million accounts [3], which represent around a fifth of the Norwegian population [4].

Twitter users can follow other users, and can be followed themselves. Unlike many other social networking sites, these relationships require no reciprocation. Also, the default option for posted tweets is to make them publicly available, though they can be restricted to the account's approved followers. Part of Twitter's appeal is, however, to share public opinions. Other social networking services like Facebook are more directed towards private interactions. Thus, according to Mislove et al. [2011], over 91% of Twitter users chose to make their communication history publicly available.

Twitter's size gives rise to a quite unique opportunity to study a decent fraction of the population. In fact, researchers have started to use tweets to measure and predict real-world phenomena, such as movie box office returns [Asur and Huberman, 2010], outcome of elections [O'Connor et al., 2010, Tumasjan et al., 2010], and stock markets [Bollen et al., 2011, Zhang et al., 2011]. While these studies shows various promise, they do not discuss to what extent Twitter accounts can be used as a

[1]Through the thesis, tweets can both refer to the text and the text including meta information.
[2]Twitter company facts: https://about.twitter.com/company
[3]http://ipsos-mmi.no/some-tracker
[4]Norwegian population https://www.ssb.no/befolkning/nokkeltall

representative sample of the population. Mislove et al. [2011] addressed this issue and found that in general, the U.S. Twitter population was a highly non-uniform sample of the U.S. population. Knowledge of these biases is important if Twitter is going to be used for population analytics.

Though Twitter enables the collection of publicly available data, there are some limitations. Twitter's *Firehose* [5] enables streaming of all public statuses, but requires special permissions. Few applications requires this level of access, and the smaller version [6], sometimes called *Gardenhose*, is sufficient for the collection of Norwegian tweets.

In this thesis, Twitter was not used for population analysis. Instead, tools were created to enable the analysis of individual Twitter accounts. More precisely, classifiers were made for predicting the gender of Norwegian users account. The focus was mainly on four areas: tweets (texts), user descriptions, profile images, and friends of the accounts. These were first investigated separately, and later combined into classifiers in attempts to increase the accuracy of predictions.

In many ways, this thesis can be viewed as an investigation of which Twitter data contains information about gender. As new information sources were gradually explored, also new, and often better, methods for analyzing the data were investigated. Some of these new methods were then applied on information previously analyzed, making some choices of analysis seem poorly motivated. Also, some analysis used in the beginning, was later dropped or removed in preference of better measures.

The datasets used in this thesis were mainly collected by Intelligent Communication AS [7]. They were, for the most part, gathered through the public stream of tweets. Therefore, the main part of the analysis is concerned with the information available thorough single tweets, and only a small part discuss information available through aggregation of multiple tweets.

1.1 Notation

I have tried to follow most conventions when it comes to notation, and be consistent throughout the report. Stochastic variables have no particular notation, but is should usually follow from the context. Note the following:
$\mathbf{x}$ is a vector.
x_i is an element in $\mathbf{x}$.
$\mathbf{x}_i$ is a data point.
$P(a)$ is the probability for the incident a.

[5]Twitter Firehose: `https://dev.twitter.com/streaming/firehose`
[6]Twitter public streams: `https://dev.twitter.com/streaming/public`
[7]Intelligent Communication AS: `http://intelcom.no/`

Chapter 2

The data

A couple of different datasets were used in this project. Mainly, tweets were gathered through the streaming APIs (Gardenhose), filtered on Norwegian tweets. This gave access to the tweets with some additional user- and meta information. Each tweet also included a link to the account's profile image, but not the actual image. Thus, images had to be collected separately. The tweets did not contain a list of the users' friends (accounts the user is following). This had to be collected through Twitter's REST APIs [1]. There are, however, some limitations on gathering the friend's IDs. Only 15 requests can be made per 15 minutes [2]. This resulted in some limitations of the size of this dataset.

When collected, the goal was to obtain datasets of tweets written in Norwegian. This was done by filtering on the 500 most common words in the Norwegian language [3]. Most of the tweets are written in Norwegian, but some are written in other languages, like Swedish, Danish and English.

The following list shows the information used in the thesis:

- **hashtags:** list of hashtags used in the tweet
- **urls:** URLs used in the tweet
- **user_mentions:** users mentioned in a tweet
- **text:** the actual tweet
- **user:**
  - **description:** a description of the user, made by the user
  - **favourites_count:** number of tweets the account has favorited
  - **followers_count:** number of users following this account
  - **friends_count:** number of users this account is following

[1] Twitter REST APIs: `https://dev.twitter.com/rest/reference/get/friends/ids`

[2] Twitter REST limitatios: `https://dev.twitter.com/rest/public/rate-limits`

[3] `https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/Norwegian_Bokm%C3%A5l_wordlist`

listed_count: number of public lists this user is a member of

name: name of user, defined by user

profile_image_url: URL to to user's avatar image

statuses_count: number of tweets (including retweets) posted by the user

friends_ids: list of users the accout is following, in the form of user IDs

More information can be found on Twitter's development web pages [4].

2.0.1 Labeling data

Classifiers are supervised methods and therefore require labeled data. Gender is not public information on Twitter, and manual labeling of tweets would be very time consuming. We therefore propose an alternative method for assigning labels to the accounts.

As shown in the list above, the names are part of the meta information available for the accounts. Thus genders can be predicted based on the names of the accounts, and labels can be assigned accordingly.

The Norwegian statistics bureau, Statistics Norway (Statistisk sentralbyrå), administers publicly available datasets of names in the Norwegian population of 2014 [5]. Names with less than 200 occurrences are excluded. According to their data, there are 2 262 660 men distributed among 801 first names, and 2 205 666 women distributed among 916 first names. Only five names are used for both males and females. They are shown in Figure 2.1, where the y-axis gives the number of individuals. These names where excluded from the labeling scheme.

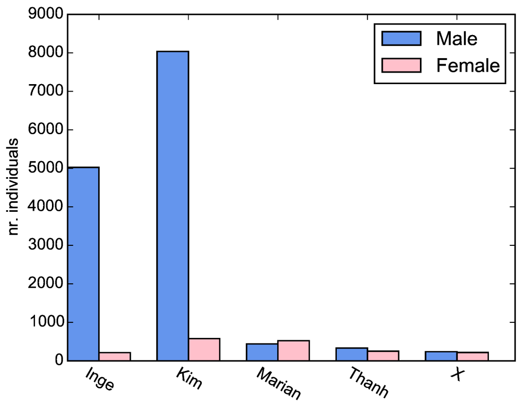


Figure 2.1: Names in the Norwegian population that are both female and male.

[4] Twitter API objects: `https://dev.twitter.com/overview/api`

[5] Statistics Norway:
`https://www.ssb.no/en/befolkning/statistikker/navn/aar/2015-01-27`

Roughly a third of the tweets collected have names that can be found in Statistics Norway's datasets. Training and test sets were made from this subset, and the rest of the tweets were discarded. This does, however, introduce some bias. Also, note that the name of an account is a very valuable feature, but is not used for obvious reasons. Therefore, if the bias is small, the test accuracy reported might actually be too low. As no test set was created from the whole population of tweets, the effect of the bias was never tested.

Twitter users choose their own name, so a male can choose a female name and vice versa. To verify our labeling approach, we looked through a subset of the data. Investigating 100 tweets with real names, there were no cases the name suggested a different gender than what we would assign to the account. In a 2012 blog post [6], Twitter reported they were able to predict gender with "more than 90% accuracy". We interpret this as not much more than 90%. Assuming this benchmark is be hard to beat, our method for labeling seems fine.

There are of course some issues with how to handle gender identities, but again assuming a minimum error rate of 10%, identity issues will hopefully be negligible.

2.0.2 Different datasets

In our datasets, some users are responsible for multiple tweets. The approximately 118 000 tweets are posted by around 35 000 users. Investigation of features specific to users, limits the data to a subset of unique users. If not, duplicate data points would exist, giving unreasonable high weight to some accounts. In addition, a classifier trained and tested on some of the same data will report unreasonable high accuracy. Classifiers fitted to features specific for the tweet, on the other hand, can use the whole dataset.

Working with profile images and friends also requires unique users. Friends comes from a different dataset, not fully contained in the first data. This needed to be addressed when classifiers were combined.

The accuracy of a classifier is very dependent on the size of the training set. A simple classifier fitted to a large dataset often outperform a better classifier with scarce data. To get the best classifier possible, we work with as large datasets as possible. This can be restricted by the computational power, or the amount of data available for the task. No distributed computing was used for the project.

[6]Twitter blog post:
https://blog.twitter.com/2012/gender-targeting-for-promoted-products-now-available

Chapter 3

Features from meta informaiton

As described in Chapter 2, each tweet contains more information than just the text. Some of this information will be discussed here. First, some handmade features were analyzed, and further, features were created from hashtags. There is some analysis of the accounts' profile colors in Appendix F, but the work was cut short in preference of other more promising areas.

3.1 Handmade features from information

As a start, simple features were derived from some of the information included in the tweets. The following covariates were made:

- length of the text
- number of hashtags
- number of URLs
- number of users mentioned in the tweet
- length of the description
- favorites_count
- followers_count
- friends_count
- listed_count
- statuses_count

The five last features (with underscores) are elements directly available from the tweets (see Chapter 2). As the features contain some user information as well as information about the tweets, a subset containing unique users was created. This was done by only including the oldest tweet of each user. The subset contained 35 244 data points, and was split in a training set of 23 614 tweets and test set of 11 630 tweets. All through the project, a relation of 2:1 was used between the training and test sets. This consistency is mainly for ease of interpretation for the reader.

3.1.1 The classifiers

Through the project, two classifiers were used: *logistic regression* and *random forests*. As many areas were covered through the thesis, the number of classifiers were limited to two, one linear and one non-linear. In general, the quality of the features is usually more important than the choice of classifier, so if these classifiers do not give decent results, it is not that likely that other classifiers would be much better. Both logistic regression and random forests are well known classifiers, and both are quite simple to tune. In addition, they give estimates of the posterior class probabilities, which is useful when the classifiers later are combined. When logistic regression is mentioned through the thesis, it is always penalized by the L1 or L2 norm. For an explanation of logistic regression and random forests see Appendix A and B respectively.

Initial experiments showed that the classifiers had a tendency to favor one class over the other. This was partly a result of unbalanced datasets. Usually the training sets consists of 60% males and 40% females. To alleviate this issue, the data was reweighted according to the inverse of the class proportions (see Appendix A.2 and B.3.3).

At the end of the project, different classifiers were combined to create more accurate classifiers. As the different datasets used have varying proportions of the genders, the individual prior probabilities did not necessarily coincide with the proportions in the test set. As classifiers were reweighed to simulate equal class proportions, this issue became more manageable.

3.1.2 Performance analysis

First, a random forests classifier was fitted to the features. The classifier was tuned using cross-validation, and we will refrain from going into further details concerning this rather broad subject.

Classification table

Table 3.1 shows the test results of the classifier. The three rows give results for females, males, and the total scores.

To be able to discuss the results, some terms need to be introduced. Consider the female row as an example. Then a *true positive* is a female classified as female, a *false positive* is a male classified as female, a *false negative* is a female classified as male, and a *true negative* is a male classified as male. Let t_p, f_p, f_n, and t_n denote the number of true positives, false positives, false negatives, and true negatives respectively. The *precision* is the ratio of true positives over classified positives,

$$\text{precision} = \frac{t_p}{t_p + f_p}. \tag{3.1}$$

The *recall* is the ratio of true positives over actual positives,

$$\text{recall} = \frac{t_p}{t_p + f_n}. \tag{3.2}$$

Table 3.1: R.F. with 200 trees on handmade features.

	precision	recall	f1-score	prop.
f	0.6	0.44	0.51	0.41
m	0.67	0.8	0.73	0.59
total	0.64	0.65	0.64	11630

So to interpret the precision and recall in the female row, the classifier manage to classify 44% of the females correctly, and 60% of the users classified as female were actually female. It would be easy to get a recall of 1.00 by classifying all users as female, but the precision would then be quite low. It is therefore common to take both into consideration when evaluating the performance of a classifier.

The total recall (third row) gives the class proportion weighted average of the scores. Thus the recall gives the accuracy,

$$\text{accuracy} = \frac{t_p + t_n}{t_p + f_p + t_n + f_n}, \tag{3.3}$$

which is the number of correct classifications over number of data points. This is a very common measure for the performance of a classifier. The total precision score is not as interpretable, and should just be considered the class proportion weighted average precision.

The prop. column in Table 3.1 gives the proportion of females and males in the test set. The total prop. gives the number of data points in the test set. Using both proportions and a count in the same columns is deliberately confusing. However, it will be useful to have both proportions and size of test set through the thesis. They are in the same columns just to make the tables more compact.

The f1-score (also called f-score and f-measure) tries to incorporate both precision and recall into one score. It is defined as the harmonic mean of the precision and recall,

$$f1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{3.4}$$

and takes values from zero to one. The total f1-score is a quite common alternative to the accuracy, and they are often quite similar. However, as Powers [2011] explains, both accuracy and f1 is biased. He therefore suggested alternatives like Phi coefficient, Matthews correlation coefficient, Informedness and Cohen's kappa. With this in mind, we still report accuracy and f1-score as they are quite interpretable and commonly used.

Now, analyzing Table 3.1, the classifier is only able to achieve f1-score of 0.64 and accuracy of 0.65. Comparing this to a classifier randomly guessing only based on the proportions (flipping a coin based on prior probabilities), it would have precisions, recalls and f1-scores approximately equal to the class proportions. This gives a total precision, accuracy and f1-score approximately equal to the squared sum of the proportions, $0.59^2 + 0.41^2 = 0.52$. A classifier assuming equal prior probabilities will have recalls and accuracy of 0.5, and precision approximately equal to the class proportions. Thus, though the random forests classifier is not particularly accurate, it is a lot better than random guessing. This implies that there is some information in the features.

Feature bar charts with class proportions

In an attempt to investigate the features in a more intuitive matter, bar charts were created for each individual feature. This is shown in Figure 3.1. The charts display the proportion of females in each bin,

$$\frac{\text{nr. females}}{\text{nr. females} + \text{nr. males}}. \tag{3.5}$$

Asymptotically, this should approach the conditional probabilities $P(\text{female} \mid \text{bin})$. If both nr. females and nr. males are zero, the charts show 0.5. Note that the bars are affected by the gender proportions in the data set. That is why most bars are around 0.4.

Each blue line shows a histogram of instances, i.e. nr. females + nr. males, in thousands. It is included to help determine if high and low values are caused by lack of data rather than differences between the genders. The y-axes are not labeled here, as considered necessary to fit the large figure on a single page. They are in general labeled though. The x-axis is limited, excluding some of the observations. Extreme values are not interesting in this analysis, and they limit the resolution of the charts.

Figure 3.1 shows very little difference between males and females, suggesting that a linear combination of the features gives little information about gender. This implies that the random forests classifier was able to find information through interactions of the features. Interestingly, the histogram of the text length seems to oscillate. The period is around 5, so maybe a sentence is commonly around five words long.

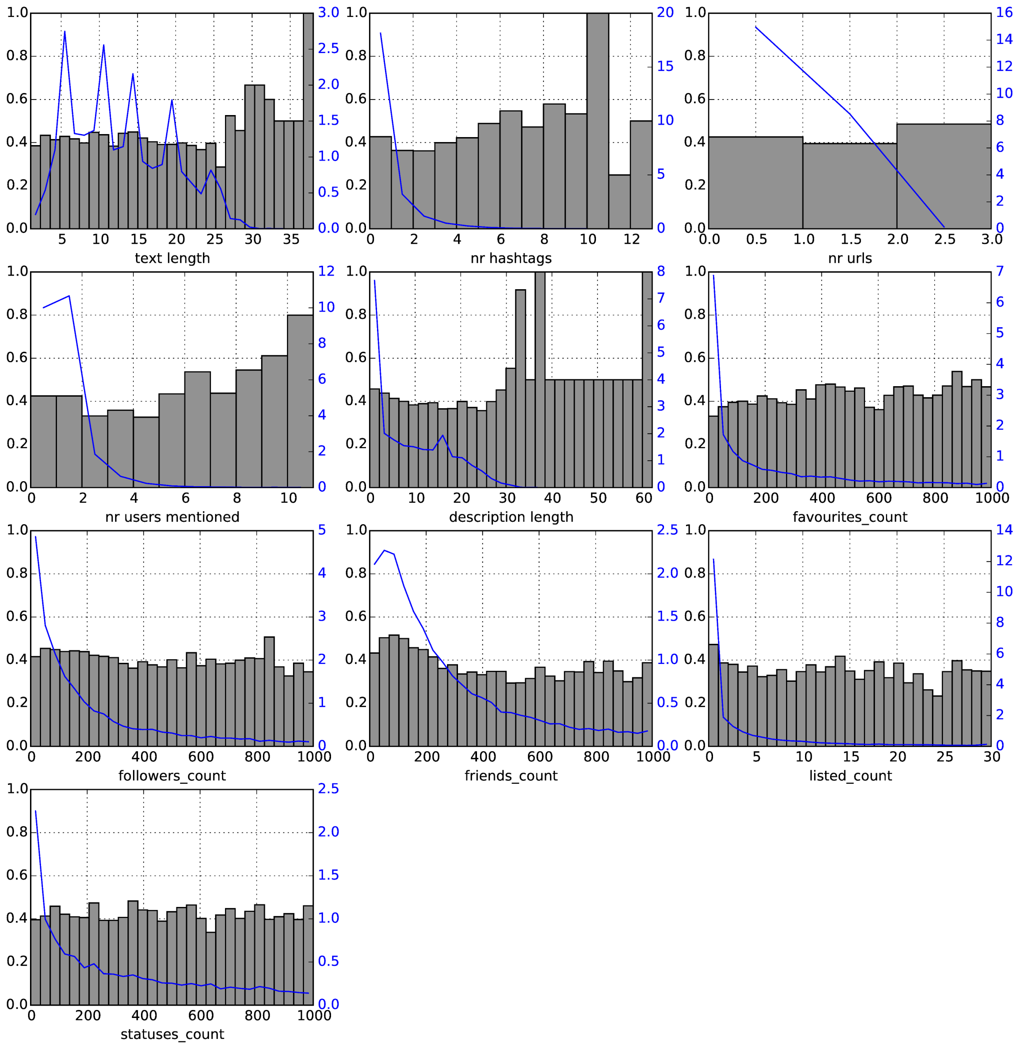


Figure 3.1: Bar charts showing proportion of females for the different covariates. The charts were made from the training set. The blue lines show number of instances for each bin (in thousands).

Probability estimates

Next, the classifier's ability to estimate the posterior class probabilities was investigated. Even though the classifier in Table 3.1 had quite low overall scores, it could potentially give decent predictions for a small subset of the data. Figure 3.2 shows the estimates for the test set. The x-axis gives the probability estimates for females in the test set, while the bars give the proportions of females within the bars. As before, the blue line shows the histogram in thousands. The red line is a reference line with slope 1 and intercept 0. The probability estimates for males are just $1 - \hat{P}(\text{female} \mid \mathbf{x})$, or 1 minus the posterior probability estimates for females. Thus, there is no reason to include a corresponding figure for male estimates.

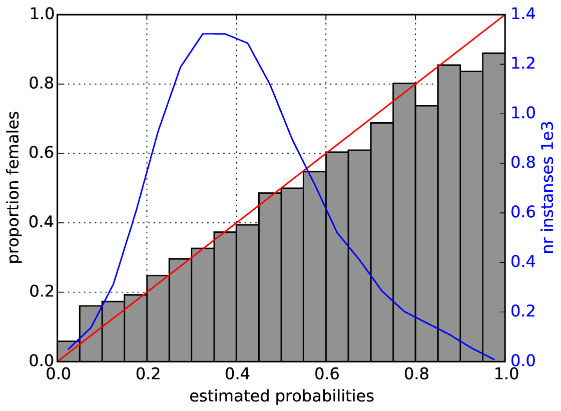


Figure 3.2: Estimates of posterior probabilities from R.F. classifier on the handmade features. The same classifier as in Table 3.1. The red line is a reference line and the bars give the proportions of females in the test set.

Investigating the figure, the probability estimates are really quite decent. The blue line gives a clear indication that most of the tweets have probability around 0.4, but still, there is definitely some information here. So while it is not a good classifier by it self, it might be able to boost the performance of other classifiers.

Even though Figure 3.1 indicated that a linear decision boundary should not be a very good fit, the experiment was repeated with a logistic regression. The classifier was tuned with both the L1 and L2 penalty. The data was standardized with zero mean and unit variance, as this is important when penalizing the size of the coefficients (see Appendix D.2). The results of the best performing classifier are shown in Table 3.2. Compared to the random forests classifier, the recall is a lot more balanced. However, the performance is possibly no better than random guessing.

Figure 3.3 shows the estimated posterior probabilities. It does not look like there is much information here at all. Thus, confirming our initial suspicion.

Table 3.2: Logistic regression on handmade features.

	precision	recall	f1-score	prop.
f	0.46	0.59	0.51	0.41
m	0.64	0.51	0.57	0.59
total	0.56	0.54	0.54	11630

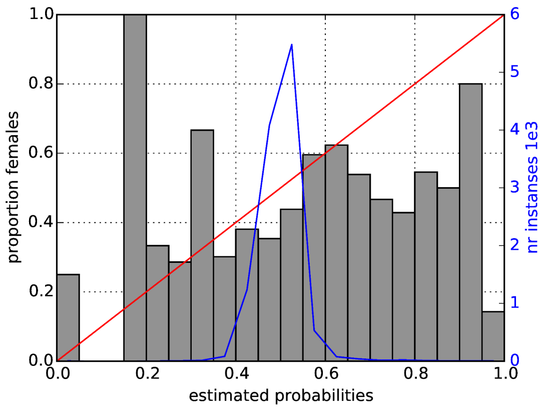


Figure 3.3: Probability estimates from logistic regression classifier on handmade features. Same classifier as in Table 3.2.

This concludes the investigation of the handmade features from the tweet information. Mostly, this was an introduction to the analytic tools used through the thesis. The struggle is to find good features for distinguishing between genders. Therefore, for the most part, a lot of different features will be investigated, but often not as thoroughly as one might find satisfactory.

3.2 Hashtags

Hashtags are a way to categorize tweets by keywords. For instance, *#2pl* is the hashtag for Premier League, created by the Norwegian TV channel TV2. The use of hashtags simplifies the process of finding tweets based on topics of interest. As hastags give the theme of the tweet, it might be a good features for separating between genders. It is important to note that tweets can be written without the use of hashtags. Actually, only around 5% of our collected tweets contain hashtags.

Every tweet collected comes with a list of hashtags used in the text (can be empty). As we are now working with tweets and not user information, a dataset containing multiple tweets per account can be used. While one might want to argue that this induce some bias, it was found necessary to get a sufficiently large dataset. The experiments in this section were tested on smaller datasets, but the results were excluded as they did not give much information. The training and test set used in this section had approximately 79 000 and 39 000 tweets respectively.

As a start, hashtags used at least 40 times in the training set were investigated. Figure 3.4 shows these hastags, where the bars show the female proportions, as described in (3.5). Clearly, some of the hashtags are quite informative. Interestingly, hastags concerning sports seem to be predominantly used by males. It is also evident that hashtags seem to be a better predictor for males than females, as there are quite few hashtags with high female proportions, but many hashtags with high male proportions. However, recall that the proportions are influenced by the proportions in the dataset. The test proportions can be found in Table 3.3a, and should be approximately equal to the training proportions used in the figure.

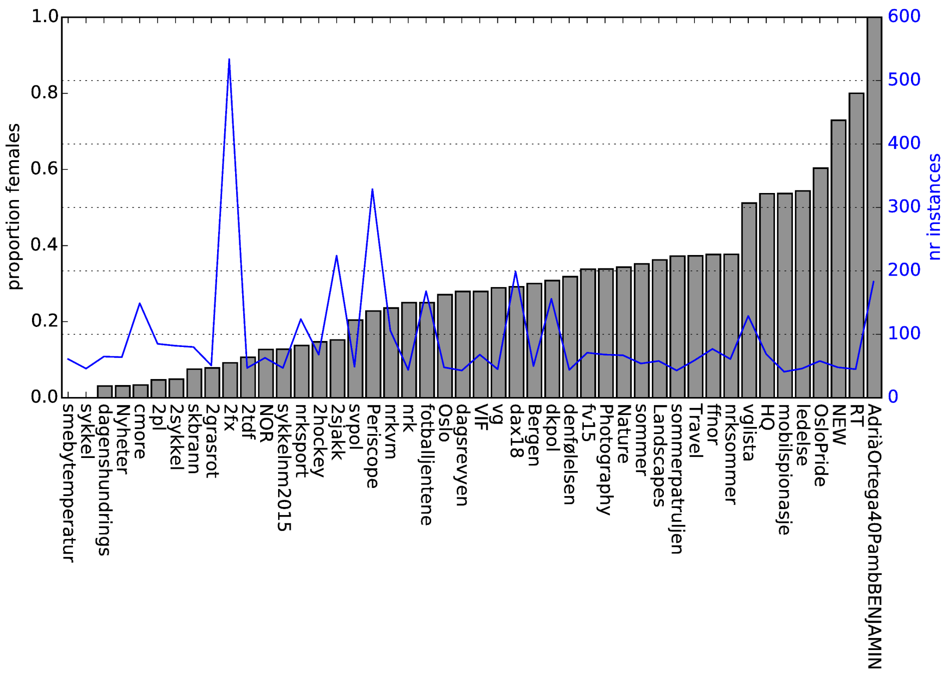


Figure 3.4: Bar chart over most used hashtags. The bars show proportion of female users, and the blue line shows number of post that contains the hashtag. If the same hastag is used multiple times in a post, they are all counted.

3.2.1 Counting features

Counting features were created from the hashtags in Figure 3.4. This gave a matrix with one row per data point (tweet) and on column per hashtag. The elements in the matrix were the number of times the hashtag was used in the tweet (0/1). A random forests classifier was fitted to the features, and the classification results are displayed in Table 3.3a. Clearly, the classifier predicts almost exclusively men. This can be explained through some investigation of the proportions. It was found that approximately 5% of the tweets in the data set used one or more of the hashtags. Thus the classifier had to choose one class for 95% of the tweets. Apparently it chose male. Approximately 72% of the tweets with the relevant hashtags were posted by males. Thus the majority here will also be predicted as male. And finally, as Figure 3.4 shows, it is easier to find good features for predicting males than females.

Obviously, hastags are not good feature for predictions on all the tweets. The classifier could, however, give decent predictions on a subset containing the hashtags. Therefore, in Table 3.3b, the experiment was repeated, but with training and test set comprised only by tweets with the relevant hashtags from Figure 3.4. The performance improved a lot, but the results for females are still not good.

Figure 3.4 did indicate a couple of very good features for separating genders, but it probably only works on a small subset. Therefore, only the features in Figure 3.4 with the highest and lowest proportion of females were used, and only tweets containing these hashtags were include in the training and test set. To get some balance, features with a lower proportion of females than 0.08 and features with a higher proportion than 0.7 where chosen. This gave the results in Table 3.3c. Clearly the scores are very good, but the test set is only around one percent of the original test set.

(a) All features. All data points in training and test set used.

	precision	recall	f1-score	prop.
f	0.72	0.01	0.03	0.37
m	0.63	1	0.77	0.63
total	0.66	0.63	0.5	38965

(b) All features. Only tweets containing relevant hashtags were used for training and testing.

	precision	recall	f1-score	prop.
f	0.72	0.36	0.48	0.28
m	0.79	0.94	0.86	0.72
total	0.77	0.78	0.75	1950

(c) Only the best separating hashtags. Only tweet containing the hashtags were used for training and testing.

	precision	recall	f1-score	prop.
f	0.95	0.9	0.93	0.28
m	0.96	0.98	0.97	0.72
total	0.96	0.96	0.96	485

Table 3.3: R.F. with 200 trees with hashtag counts as features. The hashtags are displayed in Figure 3.4.

The choice of excluding hashtags used less than 40 times was never justified. For best performance this limit should be treated as a tuning parameter. However, this was never truly tested as other approaches seemed more promising.

3.2.2 Tf-idf features

When only hastags used at least 40 times are considered as features, the subset of tweets containing the relevant hashtags becomes quite small. If all hashtags are included, the feature space becomes quite large. There are around 10 000 unique hastags in the training set. However, the features will be very sparse, and can therefore be constructed without allocating a lot of memory. If handled correctly, logistic regression can utilize this sparse structure for very fast computations. Random forests, on the other hand, can not.

Counting features are very intuitive, but not necessarily the most informative way to represent hashtags. In Appendix D.1, tf-idf is described as an alternative to counting. Tf-idf is short for "term frequency - inverse document frequency" and tries to weight the frequencies (scaled counts) by how common they are among the tweets. This is in many ways a more reasonable way to represent the hashtags, and is therefore replacing the counting features from now on.

Tf-idf features were created from the hastags, and a logistic regression was fitted to the features. Table 3.4 shows the results. Only tweets containing hashtags where included in the test set. Compared to the previous results, a much larger part of the test set was used for prediction. The total scores are not as good as in Table 3.3b, but the recall is a lot more balanced.

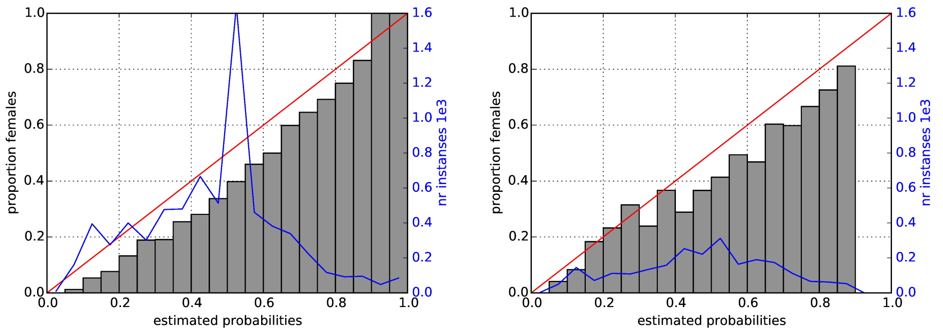
Table 3.4: Logistic regression on hashtags using tf-idf.

	precision	recall	f1-score	prop.
f	0.51	0.71	0.59	0.35
m	0.8	0.63	0.71	0.65
total	0.7	0.66	0.67	7166

As with the handmade features, the estimated posterior probabilities of the test set are interesting in the context of combining classifiers. They are therefore displayed in Figure 3.5a. Only estimates of tweets with hashtags were included. The estimates are not as accurate as for the meta information, but still decent.

When a user wants to share someone else's tweet, he or she can *retweet* it. This means that that he or she posts the same text, but with a handle to the original tweeter. This is quite common and therefore needs to be taken into consideration when making predictions. Figure 3.5b shows the estimated probabilities of only the retweets in the test set. The estimates are not quite as good as for Figure 3.5a, but they are surprisingly good. The histogram shows that there is actually less weight in the center, suggesting that it might be easier to classify retweets. One possible reason for this could be that users are more likely to retweet someone of the same gender. Another hypothesis is that users are more likely to retweet posts that include gender specific hashtags of the same gender as them selves.

The same tf-idf features were used with a random forests classifier, and the results



(a) All (b) Only retweets

Figure 3.5: Probability estimates from a logistic regression fitted to tf-idf features from hashtags. The figures only shows predictions on tweets containing hashtags. Same classifier as in Table 3.4.

are displayed in Table 3.5 and Figure 3.6. The two classifiers are actually very similar in test scores, but it looks like the probability estimates of the logistic regression are better than those of the random forests.

Table 3.5: R.F. on hashtags using tf-idf.

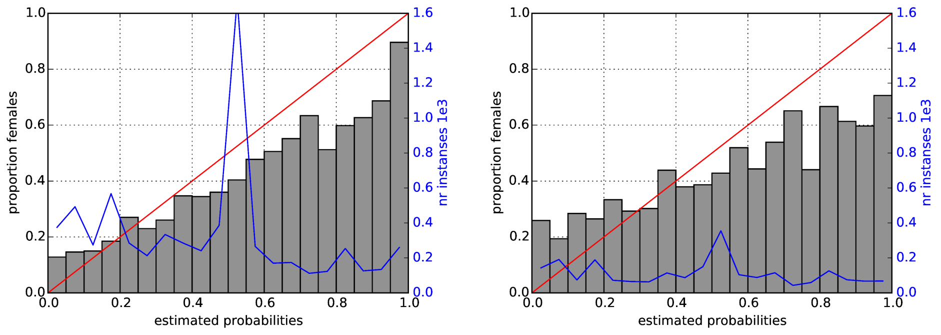
	precision	recall	f1-score	prop.
f	0.51	0.67	0.58	0.35
m	0.79	0.65	0.71	0.65
total	0.69	0.66	0.67	7166

For the counting features, the best separating features were used to investigate predictions on a small subset. This was later dropped in preference of plots of probability estimates. However, predictions on small subsets were done for parts of the thesis, but as they were very hard to compare, the results were removed.

3.2.3 N-grams

Investigating the names of the hashtags in Figure 3.4, it is clear that some of the hashtags contain the same words. One example is ”sykkel” and ”2sykkel”. This far, these similarities has not been taken advantage of. *N-grams* is a method where n consecutive characters are used as a feature, as oppose to the whole word. It is common to use n-grams in some range, e.g. 3 to 5 grams. While this gives more flexibility to the features, it also introduces new parameters that need tuning. Just as with hashtags, the grams can be represented through the tf-idf scheme. Note that n-grams can also refer to n consecutive words, but in this thesis the term is used for characters.

Table 3.6 shows the results of a logistic regression fitted to 3 to 5 grams tf-idf features. The hashtags where converted to lower case before tf-idf was applied. All



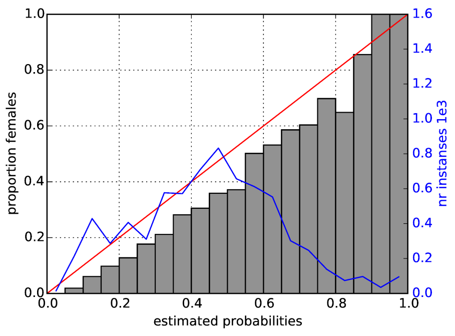
(a) All. *(b) Only retweets.*

Figure 3.6: Probability estimates from a random forests classifier fitted to tf-idf features from hashtags. The figures only shows predictions on tweets containing hashtags. Same classifier as in Table 3.5.

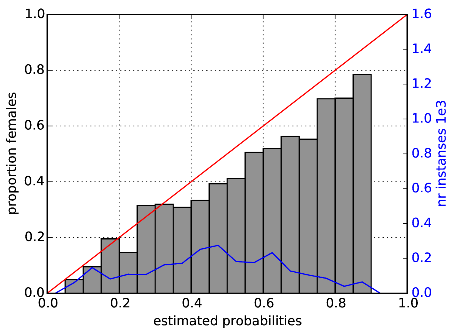
parameters were tuned through cross-validation, including the n-gram range. It seems like these n-grams result in a slightly higher accuracy and f1-score than using the hashtags in Section 3.2.2. Comparing the corresponding probability estimates in Figure 3.7 with the previous classifiers, the estimates seem to be slightly better as well. The blue histogram is wider, suggesting that this classifier was able to make predictions for more of the data. Thus is seems that the n-grams are able to capture some information that the other methods missed.

Table 3.6: Logistic regression on hashtags using tf-idf with 3 to 5 grams.

	precision	recall	f1-score	prop.
f	0.54	0.61	0.58	0.35
m	0.78	0.73	0.75	0.65
total	0.7	0.69	0.69	7166



(a) *All.*



(b) *Only retweets.*

Figure 3.7: Probability estimates from logistic regression on 3-5 grams tf-idf features from hashtags. Tweets not containing hashtags were excluded from the test set. Same classifier as in Table 3.6.

Chapter 4

Text analysis

In this chapter methods for extracting information from texts will be investigated. The texts in question are the actual tweets and the user descriptions. As with hashtags in Section 3.2, the theme of a post might be a good way to distinguish between genders. So if this can be extracted from the text, it might provide informative covariates. Also, there might be a difference between the way men and women express themselves. This can be in terms of wording, use of emojis and emoticons, or maybe something less obvious.

As hashtags were analyzed in the previous chapter, they were removed from the tweets in this analysis. This was partly because hashtags are often constructed in a way that makes them different from words, but also because results are easier to interpret when separated. It is not certain, whether or not combination of the separate classifiers gives better results than a classifier fitted to features from both hashtags and the text.

In the same way, emojis and emoticons are analysed separate from the text, and analyzed in Section 4.4.

4.1 Natural language processing

Natural language processing, or NLP, is the field of computer science, artificial intelligence, and computer linguistics concerned with the interaction between computers and human natural languages. By natural language we mean a regular human language like Norwegian and English, and not languages like programming languages. Some examples of typical NLP tasks are machine translation, optical character recognition, sentiment analysis, speech recognition, and parsing. In this thesis, however, only the field of text classification will be investigated.

There are a couple of different approaches to text classification. One method is to create handmade rules, where an expert finds some rules that are used for the classification task at hand. In our case that could be a set of rules based on words and symbols in the text. For example, if a post contains a heart next to a boys name, we classify it as female; and if there is some mention of football, we classify it as male. The accuracy of the classifier is of course very dependent on the quality of the rules. It might be quite time-consuming to create a decent classifier, if possible at all. Also, the way a language is used might change over time, requiring new sets of rules to be made at a later time.

A second approach is to create hand made features one think might be descriptive for the task at hand, and train a classifier on these features. This can for example be number of capital letter in the text, or number of exclamation points used. Again, if the features are good, than the classifier will perform well, and compared to hand made rules, this approach often requires less work. Also, even though people change the way they write, the features might still be descriptive. Thus, a new classifier can be trained on the new data. On the other hand, it might be hard to create good features, and they might be outdated with time.

The last approach is to create features based on some algorithm, and pass these features to a classifier. Note that there are algorithms that do both simultaneously. *Bag-of-words* is maybe the most straight forward method. Each text is then represented through a bag of its words. This method disregards both grammar and word order, but is, on the other hand, very simple to utilize. The possibly simplest form of bag-of-words is to create features with word counts for each word. This could be a decent approach for giving tags to documents [Yetisgen-Yildiz and Pratt, 2005]. More clever feature representations, like term frequencies and the tf-idf features discussed in Appendix D.1, are also quite common.

These sort of methods does not require human interaction. Bag-of-words are also very simple methods, enabling them to be trained on massive amounts of data. Sometimes, simpler models can outperform more complex models due to their ability to be fitted to larger datasets [Mikolov et al., 2013a].

4.1.1 Resent developments

Recent developments in text classification involve application of deep structured learning, or deep learning algorithms. These are methods with multiple non-linear layers that can potentially learn high level representations of data. Convolutional neural networks, is one such algorithm. Though originally used on images (see Appendix C.2), it has been successfully applied to text classification tasks like text categorization [Johnson and Zhang, 2014], sentiment analysis and question classification [Kim, 2014], and even sentiment analysis on tweets [dos Santos and Gatti, 2014, Severyn and Moschitti, 2015]. These papers are all based on representations of words. However, Zhang et al. [2015] recently managed to match state of the art text classifiers using characters as input to convolutional networks. This was done through transforming characters to vectors, s.t. a sentence would look like an image. This is a strong indication that language can be thought of as a signal no different from any other kind.

Another interesting development is Google's creation of *word2vec* word embeddings [Mikolov et al., 2013a,b]. These are algorithms that create mappings from words to real valued vectors. Though word embedding is not a new field, the high accuracy in word similarity tasks, and the lowered computational cost, make word2vec quite powerful.

Word2vec is quite commonly mistaken for a deep learning algorithm, but it is actually a quite shallow neural network architecture. The goal of Mikolov et al. [2013a] was actually to show that high quality word vectors could be created using very simple model architectures. By trading model complexity for lowered computational complexity, the word2vec models could be fitted to datasets of several orders of magnitude higher than previous methods.

Though word2vec has showed promising results in several NLP tasks, it is, for the moment, not that commonly used in text classification. Nevertheless, Severyn and Moschitti [2015] and Xue et al. [2014] shows high accuracy in sentiment analysis from application of word2vec on tweets and Sina Weibo (Chinese twitter).

In this thesis some handmade features were investigated, but the main focus was on methods similar to bag-or-words. More complex methods were not investigated, partly because they are harder to fit to the problems, but also because, in many text classification tasks, bag-of-words methods are still state of the art [Zhang et al., 2015].

4.2 Handmade features from text

The first approach was inspired by the email spam dataset by Lichman [2013]. Some features we thought might be informative, were hand crafted. As this is an investigation of what sort of features might contain gender information, retweets were initially excluded from the data sets. Retweets can be informative, and they are included at a later point, but it is simpler to start with just regular tweets. As there is no information in the texts specific to the users, a dataset with multiple tweets from the same accounts was used. The training and test sets contains 57 638 and 28 388 tweets respectively. Hashtags, users mentioned in the text, URLs, and locations were removed from the text. The following 18 features were extracted from the tweets:

- number of emojis
- number of emoticions
- number of words
- length of longest sequence of capital letters
- length of longest repetition of a letter
- length of longest repetition of a sign
- number of capital letters

- number of capital sequences
- number of double periods
- number of triple periods
- number of exclamation points
- number of extended letter sequences (three or more)
- number of extended sign sequences (2 or more)
- number of periods
- number of proper periods (at the end of a word followed by a space or end of line)
- number of question marks
- number of quotations
- number of stop words relative to number of words

Stop words are loosely defined as the most common words in the language [Rajaraman and Ullman, 2009]. Typically they include words like *the*, *it*, *at*, and *is* in english. There is no single universal list of Norwegian stop words, but one suggested list has been used [1]. Emojis and emoticons are explained in Section 4.4.

In Figure 4.1, the female proportions of the different features are displayed. The blue lines are histograms of instances in thousands. Clearly, it does not look like there is much information here. Nevertheless, a classifier might be able to extract information through combinations of the features. Therefore, a random forests classifier and a logistic regression were fitted to the features. The results are displayed in Table 4.1. Both perform more or less the same, though neither does a particularly good job distinguishing between the genders. Also, the logistic regression seems to have more balanced results.

(a) R.F. 500 trees.

	precision	recall	f1-score	prop.
f	0.47	0.48	0.48	0.34
m	0.72	0.72	0.72	0.66
total	0.64	0.63	0.64	28388

(b) Log. reg.

	precision	recall	f1-score	prop.
f	0.47	0.56	0.51	0.34
m	0.74	0.66	0.7	0.66
total	0.65	0.63	0.63	28388

Table 4.1: Classification results on hand made features from text.

As before, the classifiers' estimated posterior probabilities were plotted. They can be found in Figure 4.2. Clearly, neither is particularly good, though the random forests classifier seems to be able to classify men to some extent. The histogram is

[1]Stopwords: `https://github.com/Alir3z4/stop-words/blob/master/norwegian.txt`

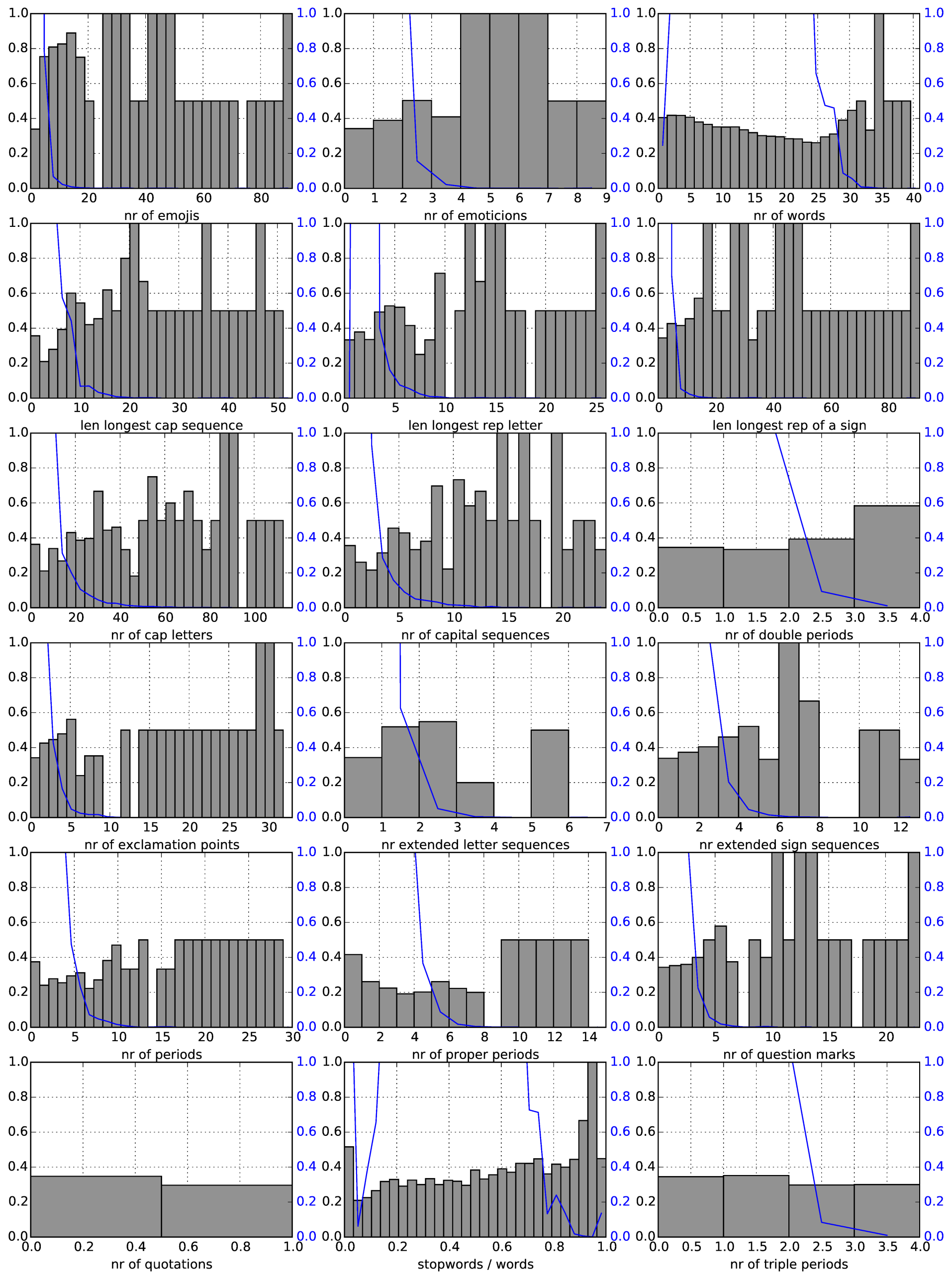
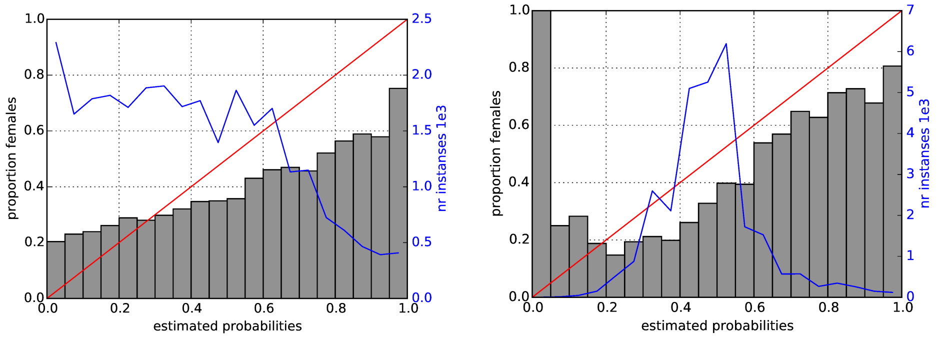


Figure 4.1: Female proportions in handmade features from the text. The blue lines show the number of instances in each bin (histograms) in thousands. The labels on the y-axes are removed due to space considerations.

quite high (2 000) for a group containing 20% women. Regardless, these features were not used any further in this project.



(a) R.F. 500 trees *(b) Log. reg.*

Figure 4.2: Estimated probabilities from classifiers on handmade text features. Same classifiers as in Table 4.1.

4.3 Words in tweets

In this section, the actual words are used as features. If some words are predominantly used by one gender, words can be quite informative. Also, a good classifier might be able to look at interactions between words. This can, however, be quite difficult. Though random forests might be able to find some interactions, the feature space will be quite large, possibly requiring massive amounts of training data.

As a start, some of the features were visualized through female proportions. Stop words was removed, as according to Rajaraman and Ullman [2009] they are often very little informative. Also, everything except words were removed, i.e. emojis, emoticons, URLs, locations, users, and hashtags. Figure 4.3 shows the 40 words with the highest proportions of female and male users (20 each). Words appearing less than 100 times in the training set were excluded. The blue line shows a histogram of number of times the words appeared in the entire training set.
The figure indicates that words could indeed give informative features, maybe particularly for males. As with the hashtags in Section 3.2, sports seems to be a good indication of a male user. Female users, on the other hand, do not appear to have any such characteristic trait.

The training set contains around 58 000 tweets, consisting of approximately 63 000 distinct words. Features were created using the tf-idf scheme and a logistic regression was fitted to them. After some tuning, it was found that features consisting of 1 to 3 consecutive words gave the best results. This resulted in over 600 000 features. Interestingly, using the L2 penalty gave better results than the L1, meaning feature

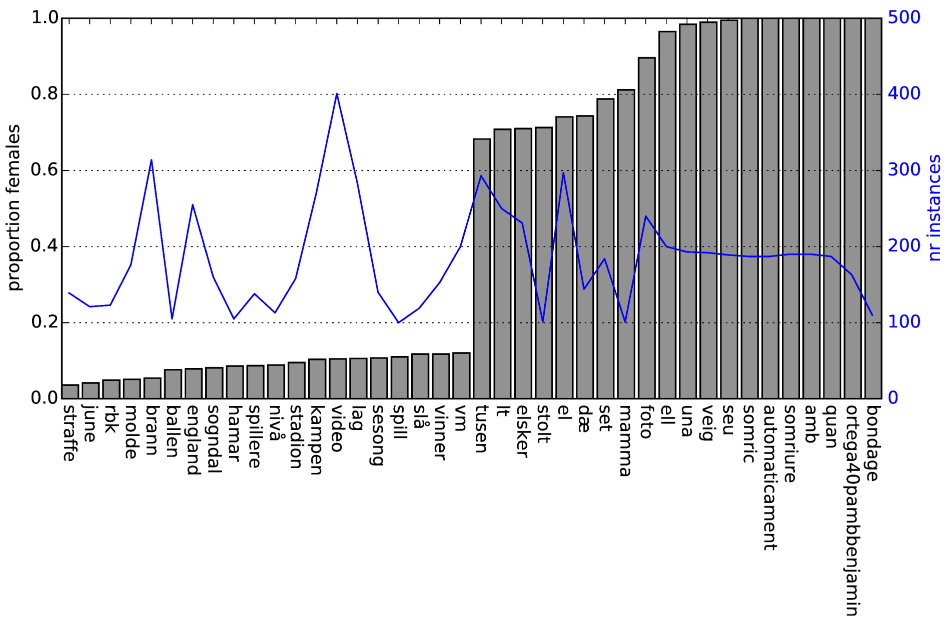


Figure 4.3: Proportion of females using words. Only words appearing at least 100 times in the training set are shown.

selection did not give good results. The coefficients were investigated and none of them vanished in the best performing classifier (can happen for very sparse features and L2 norm [Park and Hastie, 2007]).

The classification results are displayed in Table 4.2a, and it looks like this is the best classifier so far. The classifier's estimated probabilities are shown in Figure 4.4a. Though the estimates for females are in general too high, they might still be useful for combining classifiers.

(a) Log. reg. 1 to 3 words.

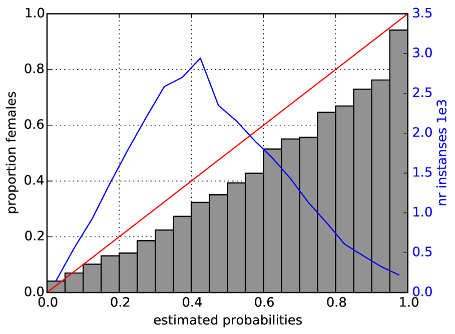
	precision	recall	f1-score	prop.
f	0.53	0.58	0.55	0.34
m	0.77	0.73	0.75	0.66
total	0.69	0.68	0.68	28388

(b) R.F. 500 trees. 1 word.

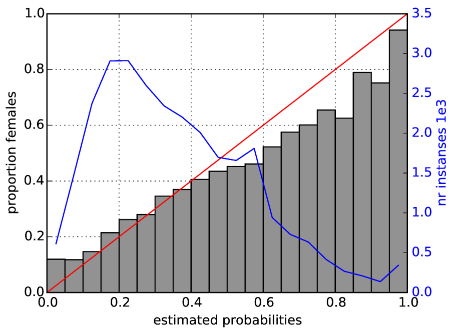
	precision	recall	f1-score	prop.
f	0.55	0.4	0.46	0.34
m	0.72	0.83	0.77	0.66
total	0.66	0.68	0.66	28388

Table 4.2: Tf-idf features on consecutive words in tweet. Retweets were removed from the datasets.

When the feature space is almost as large as the training set, a non-linear classifier is often too noisy. Methods have been proposed for applying random forests models to high dimensional data, e.g. Do et al. [2010] and Xu et al. [2012], but this will not be considered here. As the covariates are very sparse, logistic regression will have a huge computational advantage over random forests. Regardless, a random forests classifier was fitted to tf-idf features containing one word each. As Table 4.2b shows, this classifier has similar accuracy and f1-score as the logistic regression. However, it is not as balanced. The probability estimates in Figure 4.4b, shows that the random forests classifier is in general too confident in its predictions, though, they do not seem much worse than the estimates by the logistic regression.



(a) Log. reg. 1 to 3 words.



(b) R.F. 1 word.

Figure 4.4: Estimated probabilities from classifiers fitted to tf-idf feature on words in tweets. Retweets were removed. Same classifiers as in Table 4.2.

4.3.1 N-grams on text

In Section 3.2, n-grams seemed to give better features than using the whole hastags. There, the intuition was that hastags often are made up of several words, and n-grams were able to utilize this. When comparing the Norwegian language to the English language, Norwegian words are very often comprised of several words. For instance, all nouns should be written without splitting the words. Thus e.g. "gass station" will be "bensinstasjon" (gass = bensin, station = stasjon). With this in mind, n-grams can extract the individual words, which could result in more descriptive features. On the down side, the features used in Table 4.2a were made of up to three consecutive words, and this structure is lost using n-grams.

For the words features above, the logistic regression performed very similarly to the random forests classifier. As the random forests was very time-consuming, only logistic regression was used for similar text analysis in this thesis.

Table 4.3 shows the results of a logistic regression fitted to 2 to 5 grams tf-idf features. This resulted in roughly 360 000 features, a lot less than for the words. The classification performance is more or less the same as for words, but the recall is more balanced. The estimated posterior probabilities in Figure 4.5 also seem to be more or less equally good. However, the blue histogram is a bit wider, giving more confident probability estimates. So, as n-grams produce a smaller feature space with possibly better probability estimates, it is considered the better choice of features. However, the two classifiers could probably be used interchangeably.

Table 4.3: Log. reg. on tf-idf 2 to 5 grams from words in tweets. Retweets were removed from the dataset.

	precision	recall	f1-score	prop.
f	0.52	0.62	0.57	0.34
m	0.78	0.7	0.74	0.66
total	0.69	0.67	0.68	28388

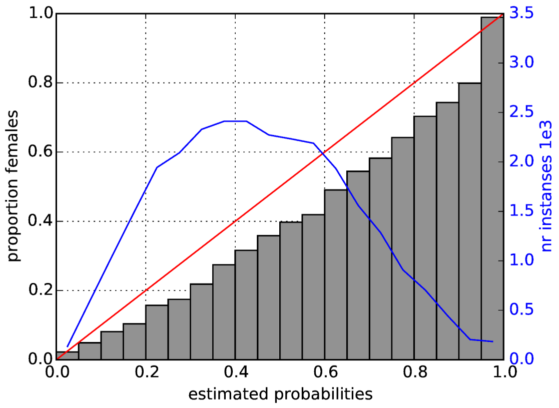


Figure 4.5: Estimated probabilities from logistic regression on 2 to 5 grams tf-idf features from words in tweets. Retweets were removed from the dataset. Same classifier as in Table 4.3.

Another option is to combine the n-grams with the words features, into one large matrix of covariates. This was briefly explored, and did not seem to affect the results. Tables and figures are excluded from the report.

4.3.2 Retweets

This far, retweets has been removed from the datasets. While this is fine to simplify the analysis, it is preferable to be able to make predictions on retweets as well. If not, retweets will be considered as missing data, which makes the combination of classifiers more difficult.

In the investigation of hashtags in Section 3.2, it was found that retweets did not require any special handling, and could be treated as regular tweets. Inspired by this, a classifier was made that did not differentiate between retweets and regular tweets. Thus, the full dataset with almost 80 000 tweets in the training set was used. A logistic regression with 2 to 5 grams tf-idf features was fitted to the tweets, and the results are displayed in Table 4.4. We see that the test set includes around 10 000 more tweets now, and the performance is not worse than without retweets in Table 4.3.

Table 4.4: 2 to 5 grams tf-idf log. reg. on text. Retweets are included.

	precision	recall	f1-score	prop.
f	0.57	0.63	0.6	0.37
m	0.77	0.73	0.75	0.63
total	0.7	0.69	0.7	38965

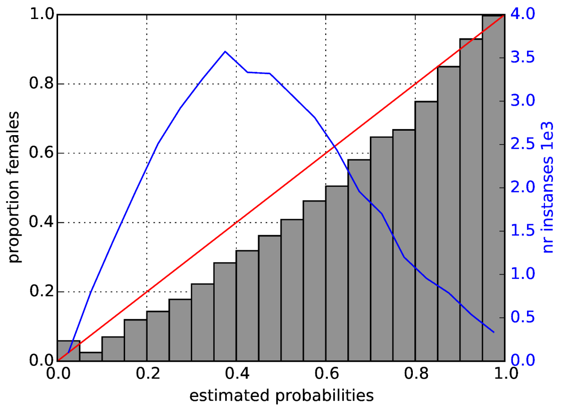
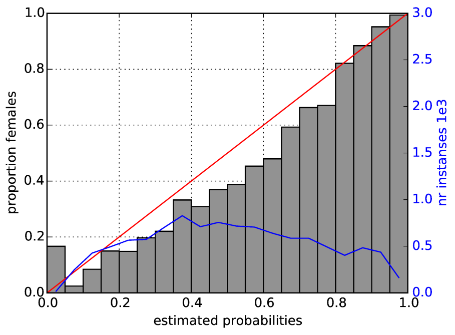


Figure 4.6: Probability estimates of log. reg. on 2 to 5 grams tf-idf features. Retweets were included in the datasets.

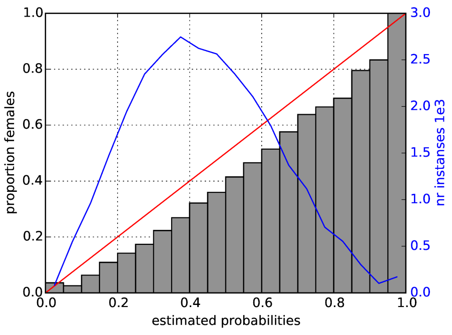
Figure 4.6 shows the classifier's probability estimates, and they actually seem better than without retweets in Figure 4.5. A similar tendency was found with the hashtags. Therefore, to further investigate the impact of retweets on the classifier, the two plots in Figure 4.7 were created. They both show posterior probability estimates from the logistic regression, but the test set was split in retweets and regular tweets. Both have quite similar quality of their estimates, but the histogram (blue line) is much wider for the retweets. This indicates that the classifier is more certain in its predictions on retweets than regular tweets. When discussing hashtags, two hypotheses were suggested to explain this. The first was that users are quite likely to retweet someone of the same gender. This does, however, not explain why the retweets are easier to predict. The other hypothesis was that very gender specific tweets are more likely to be retweeted by someone of the same gender. This is quite difficult to test, as there is no definition of what is considered gender specific.

4.3.3 Aggregtion of text

This far, all the text analysis has been done using one tweet from each account. The information available is therefore quite limited. In Appendix E, it was shown that by collecting all tweets posted by a user, the accuracy could be increased. This was, however, not thoroughly analyzed.



(a) *Only retweets.*



(b) *No retweets.*

Figure 4.7: Estimated probabilities from Figure 4.6. The two figures show exclusively retweets and not retweets.

4.4 Emojis and emoticons

Emojis and emoticons are ideograms used as part of a text, often to express emotions. Emojis are Unicode characters that represent images. A subset of the emojis available on twitter are displayed in Figure 4.8. Emoticons, on the other hand, are created from regular ASCII characters. Some examples are :) :p ;-) :3. These are, however, often replaced by an image similar to the emojis when displayed.

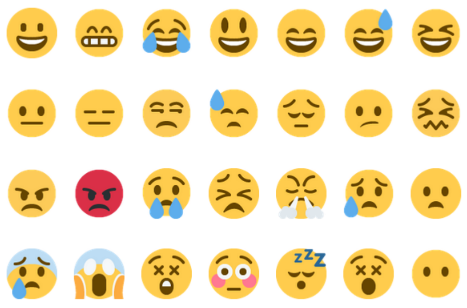


Figure 4.8: Example of emojis available on Twitter.

In the training set of approximately 80 000 tweets, almost 12 000 contains emojis or emoticons. Accordingly, classifiers based on emojis and emoticons will only be able to make predictions on this subset of the tweets. Clearly, the overall performance is not that interesting, and results will therefore only be reported on test sets where all tweets contain emojis or emoticons.

As the removal of retweets has not been necessary this far, it is assumed to not make an impact here either. Hence, the full data set is used.

Figure 4.9 shows the female proportions of emojis used at least 50 times in the training set. If a tweet contains multiple instances of the same emoji, they are all counted. The labels on the x-axis only displays the emojis supported by our version of python's matplotlib library (v1.4.3) [Hunter, 2007]. We found this more interesting than showing the Unicodes. From the figure it looks like it might be possible to classify some users as female based on the emojis alone. There are, however, seemingly no emojis that are almost exclusively used by males.

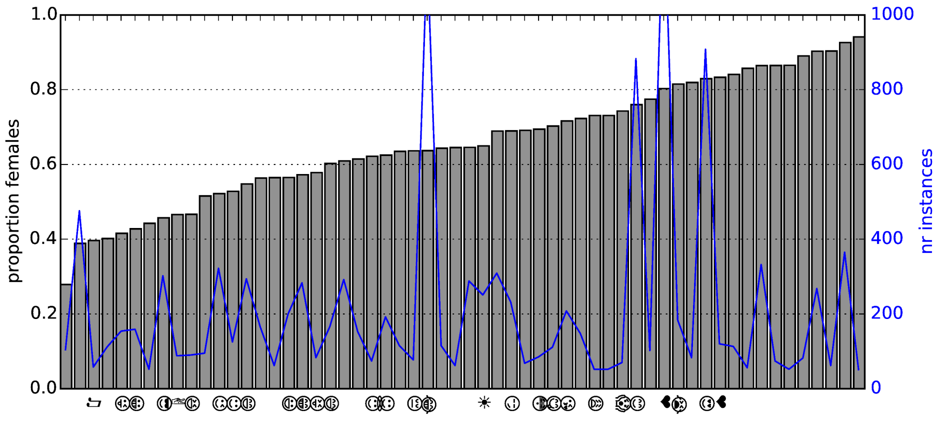


Figure 4.9: Female proportions of emojis.

Figure 4.10 shows the female proportions of emoticons used at least 20 times in the training set. Clearly a larger variety of emojis are more commonly used. Interestingly, conversely to emojis, emoticons seems to give better features for males than females. It does, nevertheless, look like the emoticons are not particularly informative.

Tf-idf features from emojis and emoticons were created and a logistic regression was fitted to them. Tuning resulted in features consisting of 1 to 2 emojis/emoticons. The training set contained both tweets with and without emojis/emoticons, but only tweets with emojis/emoticons were included in the test set. The classification results are displayed in Table 4.5. The performance is in general not very good, and the classifier clearly prefers to predict females over males. This is probably because more women use emojis. There are approximately 40% females in the training set, but the classifier reweights to get equal prior probabilities. So to the classifier, it then probably seems like almost all emojis comes from females. In addition, it is a lot easier to predict females from emojis.

The same experiment was repeated, but with the classifier trained only on tweets with emojis or emoticons. As the results in Table 4.6a show, both accuracy and f1-score increase. In addition, the recall is a lot more balanced between genders. Interestingly, features consisting of 1 to 3 emojis/emoticons gave the best score.

Considering that less than 6 000 out of the 39 000 tweets in the test set contains

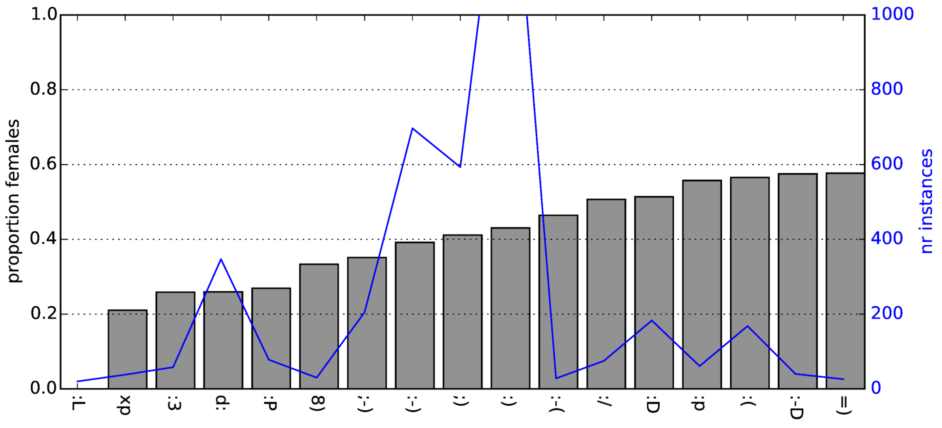


Figure 4.10: Female proportions of emoticons.

Table 4.5: Log. reg. fitted to tf-idf emojis/emoticons. Trained on full set, tested on subset containing emojis/emoticons. 1 to 2 emojis/emoticons were used.

	precision	recall	f1-score	prop.
f	0.59	0.89	0.71	0.55
m	0.65	0.24	0.35	0.45
total	0.62	0.6	0.55	5661

emojis or emoticons, and only an accuracy of 0.65 was obtained, the results are not particularly good. Hence, a random forests classifier was fitted to the data, but only 1 emoji/emoticon was used per tf-idf feature. Table 4.6b shows that there is no increase in performance.

(a) Log. reg. 1 to 3 emojis/emoticons per feature.

	precision	recall	f1-score	prop.
f	0.71	0.63	0.66	0.55
m	0.6	0.68	0.64	0.45
total	0.66	0.65	0.65	5661

(b) R.F. 500 trees. Only 1 emoji/emoticon per feature.

	precision	recall	f1-score	prop.
f	0.69	0.64	0.66	0.55
m	0.59	0.64	0.62	0.45
total	0.64	0.64	0.64	5661

Table 4.6: Classifiers fitted to tf-idf features from emojis/emoticons. Both training and test set only contains tweets with emojis/emoticons.

The posterior probability estimates were created, and are displayed in Figure 4.11. The logistic regression seems to give quite accurate estimates, but the random forests classifier does not.

In conclusion, emojis and emoticons are not considered very good features for separating genders, partly because decent predictions can only be made on a small subset of the tweets. There might, however, be some benefit from combining these classifiers with others. When combining classifiers, it is beneficial that the classifiers can make predictions on all the tweets. If not, methods for handling missing data needs to be considered. These methods can, however, be simple, like assigning the prior probabilities to tweets not containing emojis or emoticion.

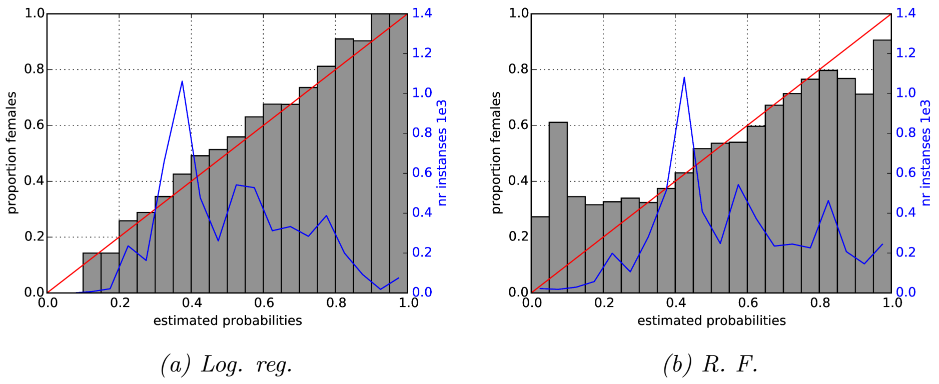


Figure 4.11: Posterior probability estimates from classifiers in Table 4.6, on emojis and emoticons. Tweets not containing emojis or emoticons were excluded from the training and test set.

4.5 User description

Each tweet obtained through Twitter's APIs contains some user information. One of the fields is the *user description* where the owner of the account can write a short text about him- or herself. This is of course completely voluntary, and around 70% of users in the datasets have written a description. Hopefully, when someone writes about them selves, they will include some clues to their gender. This can for instance be that a user describes herself as "tobarnsmor" (mother of two), or some more subtle hints, like a profession or interest that is more common among one of the genders.

During this investigation of user descriptions, a subset of the tweets with unique users was used (same as in Section 3.1). This was divided into roughly 24 000 training samples and 12 000 test samples. The task at hand is very similar to working with the actual text in a tweet. Hence, it was assumed that decisions in this section could partly be based on relevant results from Section 4.3.

Tf-idf features of 2 to 5 grams were created from the descriptions. The text was made lower case and stop words were removed. Emojis, emoticons, hashtags, etc. were not removed. Previously, these have been analyzed separately, but that was partly to get more interpretable results.

A logistic regression was fitted to the features. The results for a test set only including users with descriptions are shown in Table 4.7a. Clearly, there is quite a lot of information in the descriptions. The test scores are approximately equal to the test scores for n-grams on the tweets, though, here only on a subset.

(a) All features.

	precision	recall	f1-score	prop.
f	0.62	0.64	0.63	0.4
m	0.75	0.73	0.74	0.6
total	0.7	0.7	0.7	8791

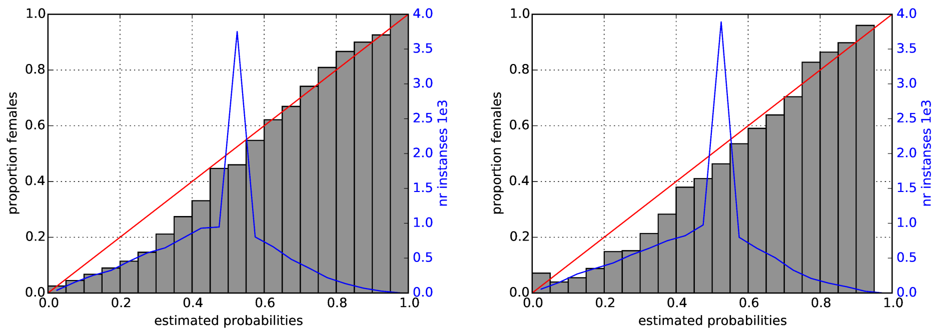
(b) Only alphanumeric features, without URLs, hashtags, etc.

	precision	recall	f1-score	prop.
f	0.61	0.64	0.62	0.4
m	0.75	0.71	0.73	0.6
total	0.69	0.69	0.69	8791

Table 4.7: Logistic regression of tf-idf 2 to 5 grams features from user descriptions. The test set only includes users with descriptions.

A different set of features were created in a similar matter, where URLs, hashtags, emojis, emoticons, users, locations, stopwords and anything else not alphanumeric was removed. The results of a logistic regression are displayed in Table 4.7b. Clearly, this classifier's performance is almost identical to the previous, and by removing all this data only 280 000 features were created, instead of the original 480 000.

The probability estimates of both models are displayed in Figure 4.12. They are both quite good, and it is hard to determine which one is better. Both feature spaces are very sparse, and the time of computations seemed to be more or less identical. When two classifiers have equal performance, it is quite common to choose the most parsimonious model.



(a) All features. *(b) Not URLs, hashtags, etc.*

Figure 4.12: Estimated posterior probabilities from description classifiers in Table 4.7. Tf-idf 2 to 5 grams. The test set contains all users, with and wihtout descriptions.

As there clearly is a lot of gender information in the description, a plot of female proportions of words is probably not particularly necessary. Regardless, a plot of the largest coefficients (absolute value) in the logistic regression in Table 4.7a was created. The largest model was chosen here to see if some of the covariates were not included in the smaller model. From the explanation of logistic regression in Appendix A, it is clear that the coefficients are proportional to the log odds. That means that a high coefficient indicates the feature is strongly associated with one class, while a very negative coefficient is associated with the other class. Note, however, that the size of coefficients is not necessarily a good measure of feature importance. A large coefficient can be associated with few data points, though penalizing the size of the coefficients to some extent restricts this.

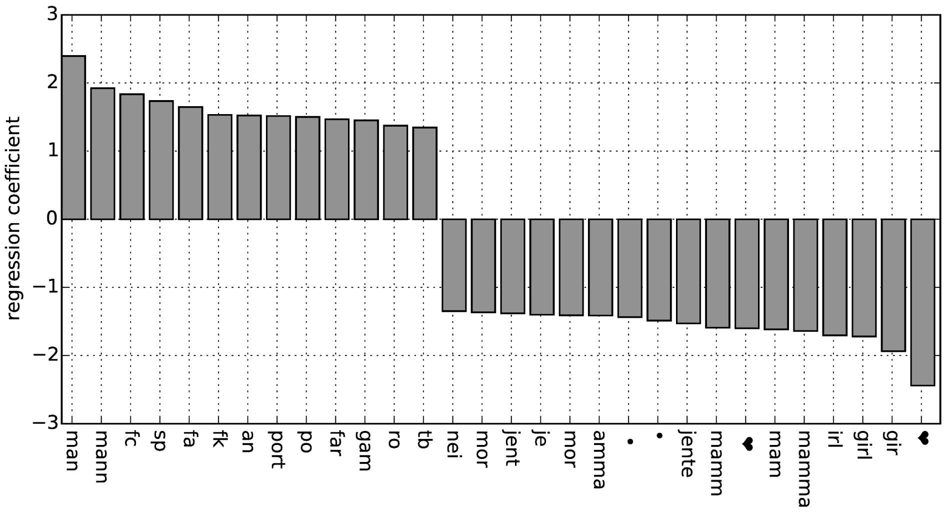


Figure 4.13: Largest and smallest coefficients from logistic regression on tf-idf 3 to 5 grams from description. Same classifier as in Table 4.7a.

The 30 largest and smallest coefficients are shown in Figure 4.13, where positive values are associated with males and negative with females. As was mentioned in the beginning of this section, we hoped that users would describe them selves using words related to gender. On the female side, we see words (and part of words) that means mother and girl, and on the male side we see the Norwegian words for man and father. In addition, there are some sports words like *fc* and *fk* (football club) on the male side.

Interestingly, there are some hearts (emojis) among the negative coefficients. Thus, the larger model includes some additional, very descriptive features. These features are probably associated with very few user descriptions, and are therefore not that important.

A random forests classifier was also fitted to the features, but the performance was, as expected, worse than the logistic regressions. Figures and tables are not included.

Chapter 5

Friends

In the introduction it was mentioned that an account on Twitter can follow other users. Following means that the owner gets updates from the users he or she is interested in. Twitter calls the accounts one follows *friends*. The friends data is not part of the dataset used this far, but was obtained through Twitter's REST APIs [1]. That gave the friends in the form of a lists of user IDs. There are some restrictions on the frequency of the requests (15 requests per 15 minuets), and the data is not collected for a very long period. Hence, the data is not as large as the previous dataset. It only consists of 9 000 accounts, divided into a training and test set of 6 000 and 3 000 accounts respectively.

Features were created from the friend IDs using tf-idf, and a logistic regression was fitted to them. Table 5.1 shows the classification results. Clearly, the friends features are very informative. The results are not quite balanced, but both precision and recall are very high compared to previous results.

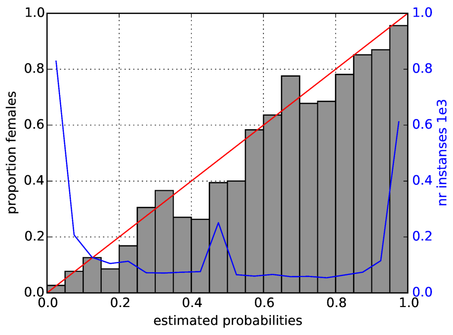
Table 5.1: Logistic regression on tf-idf features from friends.

	precision	recall	f1-score	prop.
f	0.83	0.79	0.81	0.41
m	0.86	0.89	0.88	0.59
total	0.85	0.85	0.85	3152

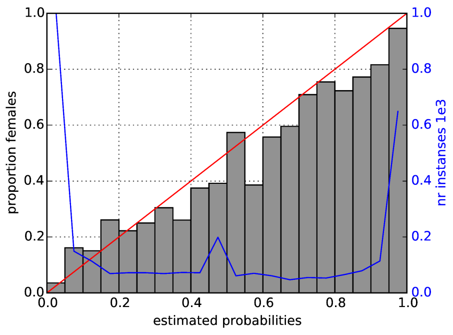
The posterior probability estimates were plotted in Figure 5.1a. Though, these are not the most accurate estimates so far, they are still quite good. Also, note how many of the estimated probabilities are either very high or very low. This is necessary if the classifier should both be able to give high accuracy and good probability estimates.

The tf-idf approach resulted in a very high-dimensional features space. The 6 000 training samples had approximately 1.3 million unique friends collectively. As

[1]Twitter REST APIs: https://dev.twitter.com/rest/reference/get/friends/ids



(a) Tf-idf.



(b) Binary.

Figure 5.1: Estimated probabilities from logistic regression on features from friends data.

previously argued, a nonlinear classifier might be too complex in this case, and a linear, like the logistic regression, is probably the better alternative. Regardless, a random forests classifier was fitted to the same features. The results in Table 5.2 reinforce our claim.

The feature space is very sparse. As before, this gives the logistic regression a huge computational advantage. While it completed almost instantly, the random forests classifier took almost an hour to train.

Table 5.2: Random forests classifier on tf-idf features from friends.

	precision	recall	f1-score	prop.
f	0.85	0.66	0.74	0.41
m	0.8	0.92	0.85	0.59
total	0.82	0.81	0.81	3152

Next, binary features were used instead of tf-idf. The results can be found in Table 5.3 and Figure 5.1b. The table is quite similar to Table 5.1, but the plots displaying the estimated probabilities are somewhat different. The binary features have more observations in the highest and lowest probability estimates. Also, the quality of the probability estimates are not very different. This suggests that the binary features might be the best approach. However, later when the probability estimates were combined with the other classifiers, the tf-idf features seemed to result in slightly higher performance.

Figure 5.2 shows some of the friends in the dataset. The blue line is a histogram, giving number of users in the training set that is following the account. Only the largest and smallest proportion of females are displayed, and features with less than 200 instances were excluded. The names corresponding to the user IDs were

Table 5.3: Logistic regression on binary features from friends.

	precision	recall	f1-score	prop.
f	0.81	0.79	0.8	0.41
m	0.86	0.87	0.86	0.59
total	0.84	0.84	0.84	3152

retrieved through the python package *python-twitter* [2]. The figure gives a good indication of why the classifiers perform so well.

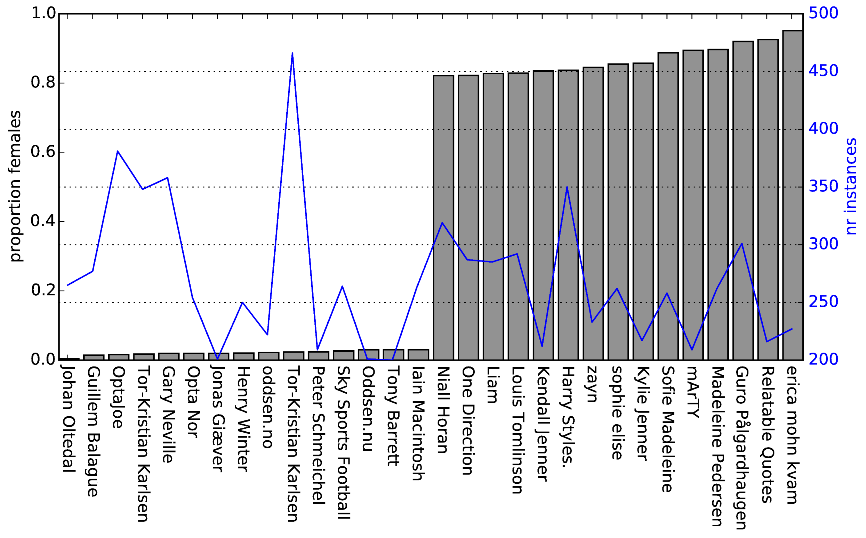


Figure 5.2: Female proportions in friend features. Only the friends with the highest and lowest proportions of females are displayed. Features with less than 200 instances in the training set were excluded from the figure.

Figure 5.3 shows the largest coefficients (absolute value) from the logistic regression on the binary features. Positive values indicate male, while negative values indicate female. Comparing the two figures, there are not a lot of the same friends displayed. The figures are not really expected to be that similar. An account that is proportionally followed by more female users, but has more followers in absolute numbers, can result in a higher reduction in the residual error. Thus resulting in a more negative coefficient. However, both figures show some of the same trend discussed earlier, where males are interested in sports.

[2]python-twitter: `https://github.com/bear/python-twitter`

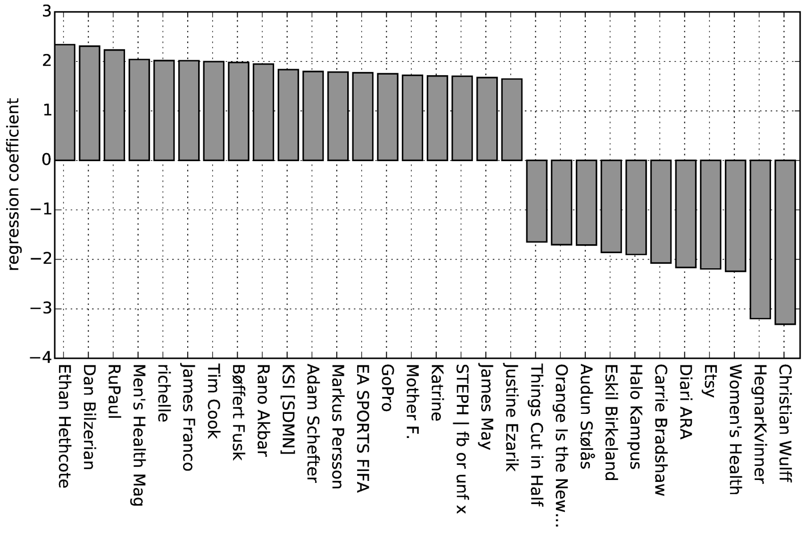


Figure 5.3: Largest coefficients (absolute value) from the logistic regression on friends with binary features. Table 5.3. Positive coefficients indicate male users, while negative indicate female.

Chapter 6

Profile images

In this chapter, the profile images of the twitter accounts are investigated. A random subset of the images is displayed in Figure 6.1. The letter above each image gives the gender of of the account. As some users have pictures that are not of themselves, but of other people or non-human objects, the expectations of the classifiers are limited. Just based on the pictures below, accuracy of 90% seems to be a rough estimate of the upper limit. For a large dataset, a classifier might be able to make decent predictions on some non-human objects (e.g. a football might indicate a male account). As the collected data was somewhat scarce, this was not expected by the classifiers in this project.

Two approaches to image classification were investigated. The first was to extract faces from the images, so pixel values would become decent covariates, while the second was to use a convolutional neural network as a feature extractor.

6.1 Image recognition

Computer vision is a field that includes methods for acquiring, processing and analyzing images. In some sense, the objective is to enable machines to duplicate the human vision. Computer vision is not only used for classification. It is also concerned with tasks like image restoration, scene reconstruction (compute 3D model of a scene), image compression, and motion analysis.

An image is commonly made of pixels. Each pixel usually consists of three numbers, one each for red, green, and blue. Thus, an image of n by m pixels is represented by a n by m by 3 array. For high-resolution images, the array can grow quite large.

For an unprocessed image, the amount of information provided by one pixel alone is minimal. Consider an image where all the pixels were presented one at a time. A human would probably have a hard time extracting much information from this representation. This is because humans do not get information from the individual pixels, but rather from combinations of neighboring pixels. The same

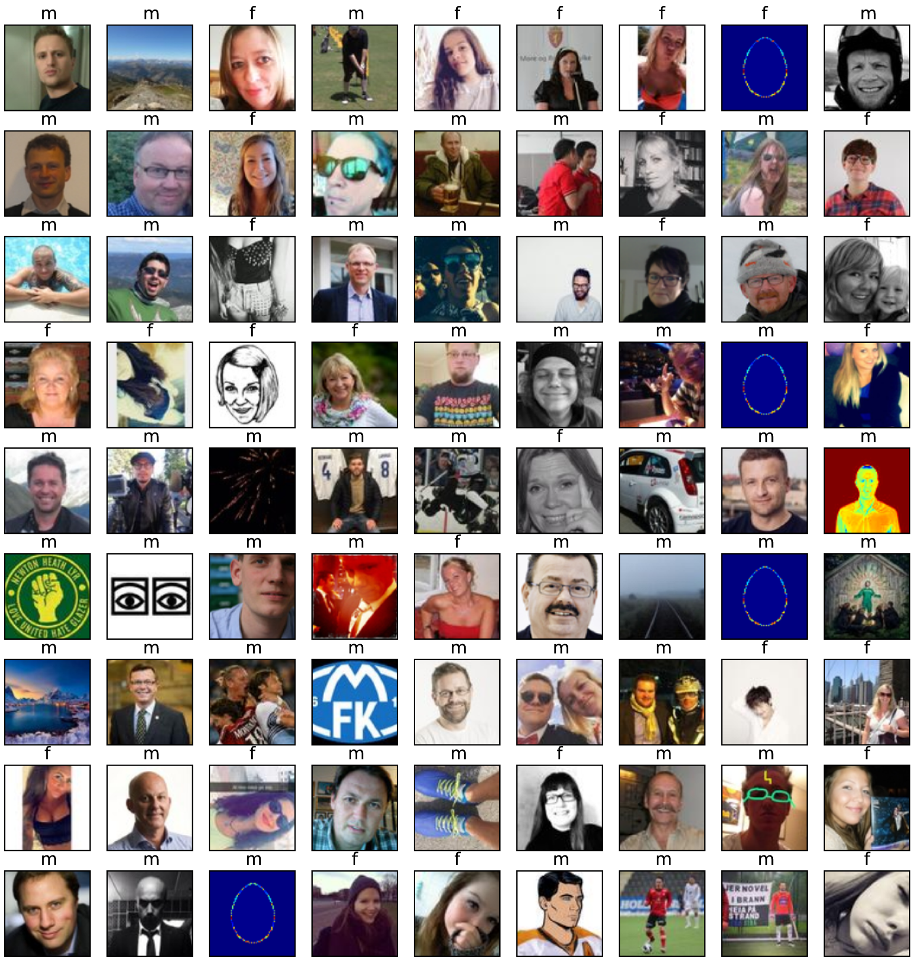


Figure 6.1: A subset of the profile images in the training set. The character above an image gives the gender of the account.

way, a classifier will have a hard time making predictions with pixels as features. Even a non-linear classifier like random forests, will not be able to combine pixels into meaningful areas. This is because random forests finds combinations between specific features, and not general rules for combinations of neighboring features. Hence, raw pixels are, in general, not considered good features for a classifier. Different approaches have be suggested to overcome this problem. The most straight forward might be to preprocess similar images. Take for instance the popular MNIST dataset [1] of handwritten digits. The digits have been size-normalized and centered in a fixed-size image. This preprocessing is done so well that the pixels start to give meaning by them selves. Lecun et al. [1998] showed that something as simple as k-nearest neighbors with euclidean distances was able to get 95% accuracy, and a linear classifier (1-layer neural net) 91.6%.

While preprocessing might work very well for digits, it is harder to apply to more complex images. Especially if the preprocessing should not have any human involvement. Therefore, as an alternative, different algorithms for extracting meaningful features from the pixels have been suggested. This framework is often divided into two parts: *detectors* and *descriptors*.

Detectors are algorithms for detecting interest points in images. Typically, the algorithms looks for:

edges: e.g. Canny [Canny, 1986] and Sobel operator [Sobel and Feldman, 1968].

corners: e.g. FAST [Rosten and Drummond, 2006] and Harris [Harris and Stephens, 1988].

blobs: e.g. LoG [Marr and Hildreth, 1980] and DoG [Bundy and Wallen, 1984].

ridges: e.g. [Haralick, 1983] and M-reps [Pizer et al., 2001].

Descriptors, on the other hand, are used to describe the patches around the interest points. These can again be used as features to a classifier. HOG by Dalal and Triggs [2005] and GLOH by Mikolajczyk and Schmid [2005] are two examples of some well-known descriptors.

Some algorithms include both the detection and description. SIFT by Lowe [1999], SURF by Bay et al. [2008] and BRISK by Leutenegger et al. [2011] is in this category. It is somewhat common that these and the other descriptors are patented.

The method of detector/descriptor pairs has been quite commonly used the last 20 to 30 years. However, recent deep learning algorithms, particularly convolutional neural networks, has been able to outperform these methods in several areas. Maybe most notably in the 2014 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC [Russakovsky et al., 2015]), where the top two submissions were convolutional nets [Szegedy et al., 2014, Simonyan and Zisserman, 2014]. However, some of the older algorithms might have a computational advantage over the deep

[1]MNIST: `http://yann.lecun.com/exdb/mnist/`

architectures, possibly making them more suitable to some tasks like real time applications.

6.2 Retrieving images

As previously mentioned, profile images were not found for all users, limiting the size of the dataset. One possible explanation for missing images might be that some users have changed their profile image after the tweets were gathered.

The images were retrieved through URLs in the user information of the collected tweets. By altering these URLs, it is possible to get larger and smaller images [2]. If an image is retrieved in its original form, it can come in a variety of sizes, some of them very large. For simplicity, the largest fixed sized images were retrieved, giving a dataset of [73 x 73 x 3] sized images. This is a quite low resolution, and comes at the cost of some information loss. While the low resolution might inhibit the classifier, investigating the images in Figure 6.1, the resolution seems to be fine in most cases.

The images were split into a training and test set of approximately 12 000 and 6 000 images respectively.

6.3 Eigenfaces

It was mentioned that working with raw pixel values is rarely a good approach for image classification. It requires that the images are very well preprocessed, such that the pixels can be compared. Also, the number of covariates will be quite high, and a lot of the pixels will have the same importance. Therefore, better approaches often involves extracting features from the pixels through some transform. Principal component analysis (PCA) is a quite common, straight forward method used for extracting informative features. It involves finding a set of vectors in the feature space, where a lot of the variation can be explained. Then, the features can be projected orthogonally onto theses vectors, reducing the dimensionality of the feature space. Hopefully, the information loss through the projection is minimal. PCA is therefore known as a information compression method.

More rigorously, PCA finds an orthogonal basis of p vectors, in which the pixels covariance matrix is diagonal. This can be done through an eigenvalue decomposition of the covariance matrix, where the (normalized) eigenvectors are the principal components (new feature directions), and the eigenvalues give the amount of the variance explained in each direction. The new features are the projection coefficients (the projection is a linear combination of the eigenvectors), when the original features are projected onto a subset of the eigenvectors. As the eigenvalues gives the variance explained, the total variance explained can be controlled through the selection of eigenvectors.

[2]Twitter profile images:
https://dev.twitter.com/overview/general/user-profile-images-and-banners

This methods is unsupervised and does not necessarily give good results, but it can potentially improve both accuracy of the classifier and time of computations. The number of principal components used is also a parameter that needs to be tuned. This could be done by not restricting the number of principal components, but let the classifier do the feature selection. However, it is a lot less computationally expensive to only create a limited amount of principal components. For more information on PCA, see e.g. Hastie et al. [2009].

For PCA to give better features than just pixel values, it is required that a large part of the information can be compressed into a few principal components. With unstructured images, this is often not the case. So, just to get a base line, a logistic regression was fitted to the 20 first principal components of the images (in gray scale). As PCA is sensitive to scaling, the pixel values were standardized.

The results can be found in Table 6.1, and they are clearly not particularly good. The estimated probabilities in Figure 6.2 are also quite uninformative. This makes sense as there is very little collective structure in the images. Some are of faces others are of bodies, multiple people, or something completely different.

Table 6.1: Logistic regression on PCA features from images.

	precision	recall	f1-score	prop.
f	0.41	0.56	0.47	0.34
m	0.72	0.58	0.64	0.66
total	0.61	0.57	0.59	6239

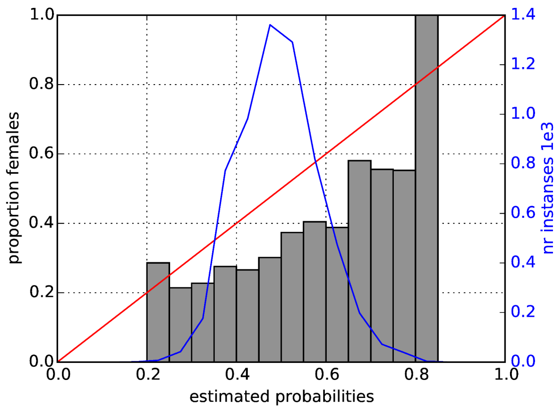


Figure 6.2: Estimated posterior probabilities from logistic regression on PCA features from images. Same classifier as in Table 6.1.

If faces could be extracted from the images, rotated so the eyes were horizontal, and the extracted face images were size-normalized, then the pixels would be easier to compare. However, face detection is probably not much easier than classifying gender. Also, the images are labeled according to gender, but there are no face labels, so these would have to be hand labeled. In conclusion, there is little point in spending time on this.

There are, however, some face detectors available through different computer vision libraries. openCV [Bradski, 2000] is one such library, providing easy implementation of face detection. The library does object detection using Haar feature-based cascade classifiers, as proposed by Viola and Jones [2001]. Haar features are just like the convolution filters described in Appendix C.2.1, and through the introduction of the *integral image*, these can be calculated extremely efficiently. An adaboost based learning algorithm [Freund and Schapire, 1997] is used to select a small number of critical visual features, and thus create a computationally efficient classifier. Finally, increasingly more complex classifier are applied in a cascade, which allows for background regions of images to be quickly discarded.

The algorithm focus primarily on computational efficiency, but Viola and Jones [2001] also report very promising accuracies. openCV provides a trained version of the classifier, but it still have some tuning parameters that need to be set. These primarily concern scaling iterations of the images, and tuning of the accepted false positive rate.

The cascade classifier was tuned and applied to the images in the datasets. If several faces were detected in one image, only the first was used. As the classifier was not able to find faces in all the images, subsets with detected faces were created for the training and test set. The extracted faces were scaled so they were all the same number of pixels [50 x 50]. Features were extracted using PCA and a logistic regression was fitted to the 100 first principal components.

The results in Table 6.2 shows a large increase in performance compared to the images in Table 6.1. However, the classifier was not even able to make predictions on half the test set, making the overall performance less impressive. Nevertheless, the performance on the subset is quite good, and can possibly be valuable if combined with results from other classifiers.

Figure 6.3 shows the estimated posterior probabilities. Though, in general, the estimates for females are too high, they might be useful.

Table 6.2: Logistic regression on PCA features from extracted faces.

	precision	recall	f1-score	prop.
f	0.63	0.78	0.7	0.33
m	0.88	0.77	0.82	0.67
total	0.8	0.77	0.78	2686

The eigenvectors obtained by application of PCA to images of faces are often referred to as *eigenfaces*. The 25 first eigenfaces from the experiment above are displayed in Figure 6.4. Clearly, the vectors resembles faces, or facial structures. The different eigenfaces represent sunglasses, tilting of the head, eyebrows, etc. The

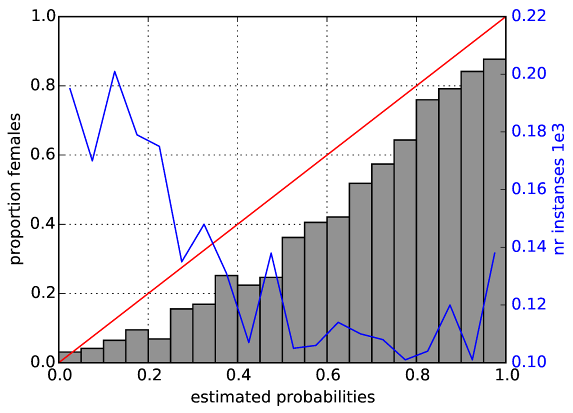


Figure 6.3: Estimates of the posterior probabilities for a logistic regression on PCA features from extracted faces. The same classifier as in Table 6.2.

projection of an image (of a face) onto the space of these eigenvectors, is then the best linear reconstruction (linear combination of the eigenfaces) of the image, under squared error loss. The coefficients of the linear combination, are the features used in the classifier. So if for instance men has thicker eyebrows, that might result in a higher coefficient for that eigenface. Thus, egenfaces can possibly provide much more informative and interpretable features than the pixels. However, though the egenfaces have a lot of potential, the method is not considered very good for image classification. This is partly due to the required preprocessing (extraction of faces).

Figure 6.5 and 6.6 show some of the males that were misclassified as female. The first figure shows the extracted faces while the other shows the full pictures. From the images in Figure 6.5 is it clear that some of the faces are tilted. While tilting of the head could be informative, it makes it harder to compare pixel values across images. If the images could be rotated, that might increase the performance of the classifier. One way to do this, is through eye detection. If the eyes are found, then the image can be rotated such that the line formed by the eyes is horizontal. openCV includes a cascade eye detector, but it was not able to find eyes in the dataset. This was probably caused by the low resolution of the images, as it seemed to work fine on some test images with higher resolutions. It might be possible to get higher resolution images from some of the users, but this was not pursued in preference of the convolutional neural networks in the next section.

Also, a as there are not that many features, a random forests classifier was fitted to the same features as the logistic regression in Table 6.2. Both the classification scores and the quality of the probability estimates were not as good as for the logistic regression. Tables and figures are not included.

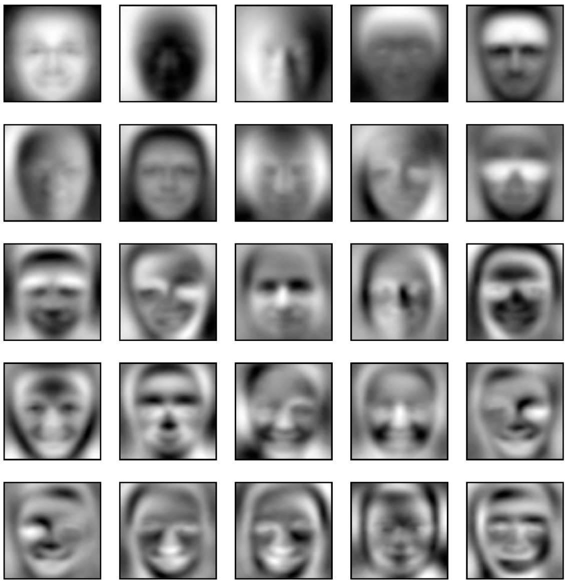


Figure 6.4: First 25 eigenfaces used by logistic regression in Table 6.2.

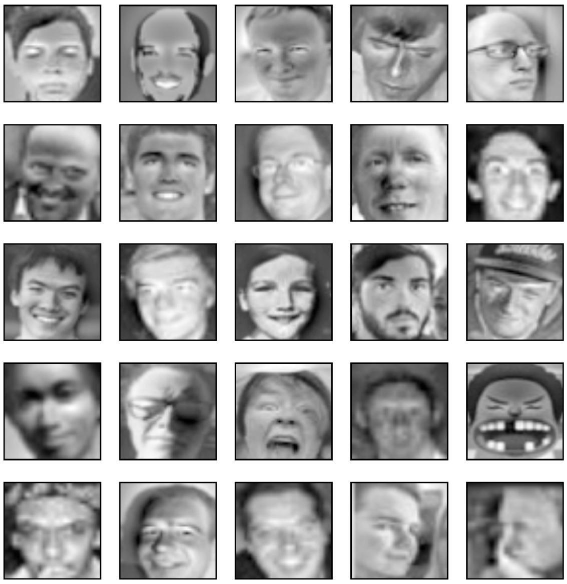


Figure 6.5: Subset of males classified as females by logistic regression in Table 6.2. These are the faces that were later passed through the PCA. The faces were extracted from images in Figure 6.6.

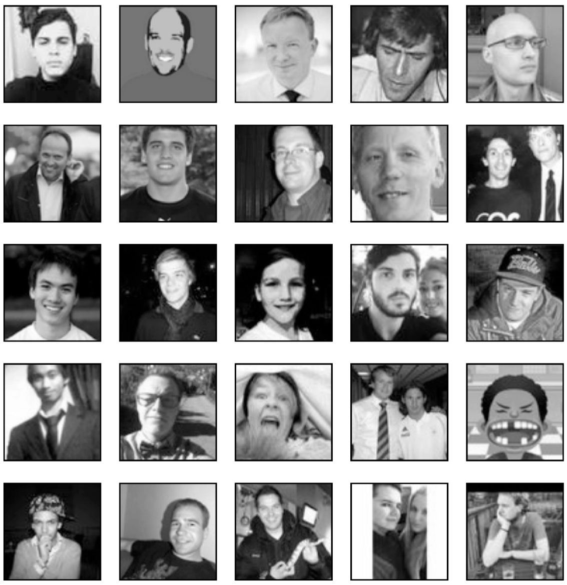


Figure 6.6: Subset of males classified as females by logistic regression in Table 6.2. These are the full images, with extracted faces in Figure 6.5.

6.4 Convolutional neural networks

Convolutional neural networks (CNNs or ConvNets) are neural networks comprised of several different layers. The layers are typically *convolutions*, *pooling*, or *fully connected*. By combining the layers in a deep architecture, a ConvNet can potentially learn quite high level features. Though ConvNets are considered supervised methods, they were applied in an unsupervised matter in this thesis. A trained net was obtained, and used as a fixed features extractor. This application of ConvNets is considered a branch of *transfer learning*. More on transfer learning can be found in Appendix C.2.6.

The ConvNet used in this thesis was a version of GoogLeNet [Szegedy et al., 2014], trained by Sergio Guadarrama [3]. As discussed in Appendix C.2.5, there are other candidates that are possibly better suited for transfer learning tasks, but Guadarrama's trained GoogLeNet was one of the best we found with an unrestricted license.

For an explanation of convolutional neural networks see Appendix C.2.

Deep learning algorithms have become increasingly popular over the last couple of years. This has resulted in numerous tools for implementations of the different algorithms. Three of the most well known are Theano by Bergstra et al. [2010], Bastien et al. [2012], Torch7 by Collobert et al. [2011], and Caffe by Jia et al. [2014].

[3]GoogLeNet Caffe: `https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet`

In this thesis, Caffe's python interface was used.

Caffe is a deep learning framework developed by Jia et al. [2014] as UC Berkeley. It provides a matlab and python interface for ease of implementation. One of Caffe's strengths is that it hosts a *Model Zoo*[4]. There users can make their trained models publicly available. As deep learning architectures can be quite computationally expensive to train, the Model Zoo can be very useful for initialization of networks and transfer learning tasks. The model Zoo includes other versions of the GoogLeNet ConvNet, but all with restricted licenses.

When a trained ConvNet is used as a fixed features extractor, it is important to consider which layer to extract from. If the new data (what we want to classify) is similar to the data the net was trained on, higher level features might be the best choice. On the other hand, if the two datasets have little in common, lower levels features will probably work better. GoogLeNet has 121 different layers. The net was trained on 1000 classes [5] mostly "everyday objects" like animals, clothing and personal objects. It is therefore assumed that higher level features would be the most informative. Features were extracted from `inception_5b/output`, as it is the highest layer with more nodes than classes. It provides 55 176 features, but they are quite sparse. Ideally, other layers should also have been tried.

The extracted features were used to train a logistic regression classifier. As the logistic regression is able to take computational advantage of a sparse structure, the features were only scaled, and not centered. The training and test set were comprised of the same images as in Section 6.2.

The classification results can be found in Table 6.3. Both accuracy and f1-score are very good, though not quite as good as for the friends data. The estimated probabilities in Figure 6.7 are also not too bad. They do not fit the line very well, but they are probably good enough to boost the results of the friends classifier.

Table 6.3: Logistic regression on output from inception_5b/output in GoogLeNet.

	precision	recall	f1-score	prop.
f	0.71	0.73	0.72	0.34
m	0.86	0.85	0.85	0.66
total	0.81	0.81	0.81	6239

While accuracy of 0.81 is quite good, it was mentioned that a human should be able to get around 0.90. So to see how well the classifier compared to the human eye, some of the misclassified images were displayed in Figure 6.8 and 6.9. The first figure shows males classified as females, and a couple of images are actually of females. The majority of the images in this figure were quite understandably hard for the classifier to get right, but there are some male, bearded faces that should be quite easy. The second figure shows females classified as males, and also here it is, for the most part, understandable that the classifier had a hard time. Though there are some images that are clearly female.

[4]Caffe Model Zoo: http://caffe.berkeleyvision.org/model_zoo.html

[5]ILSVRC2014 classes: http://image-net.org/challenges/LSVRC/2014/browse-synsets

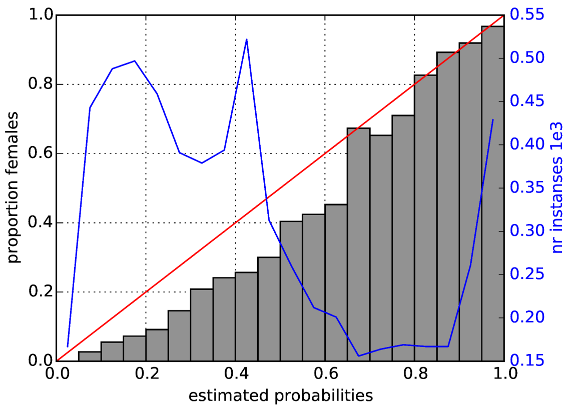


Figure 6.7: Estimated probabilities from logistic classifier on inception_5b/output layer in GoogLeNet.

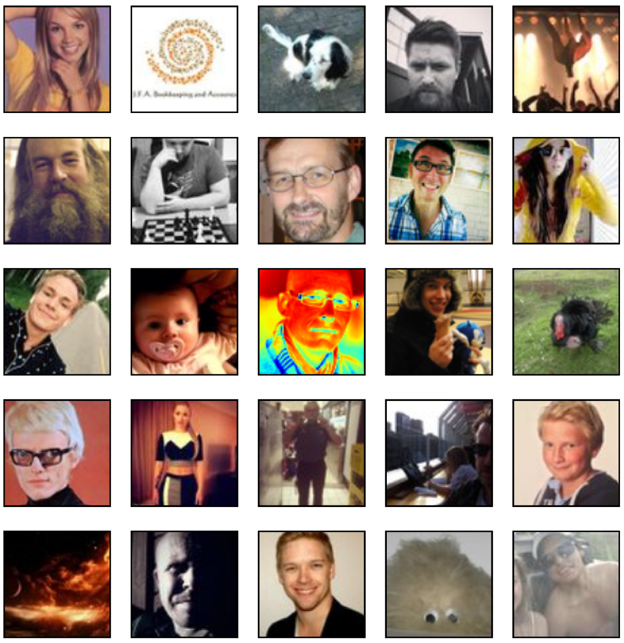


Figure 6.8: Subset of males misclassified as females from Table 6.3.

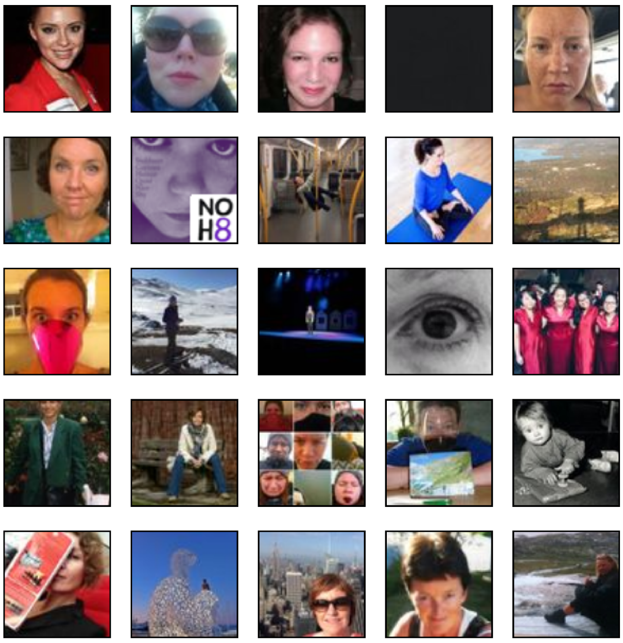


Figure 6.9: Subset of females misclassified as males from Table 6.3.

We therefore conclude that the classifier does a good job, but it should be possible to get a bit higher accuracy. This could possibly be achieved by training the ConvNet on the profile images, or another dataset of gender labeled images.

Caffe can utilize a GPU if available, which can result in a very large speedup compared to a CPU. In this thesis, only the CPU implementations were used. The forward pass through the network was very slow, potentially restricting the areas this method can be applied.

Chapter 7

Combining classifiers

In this chapter, some of the previous classifiers, now referred to as *base classifiers*, will be combined in an attempt to boost their performance. The end results are highly dependent on the correlation between the classifiers. In some ways, we would want the classifiers to be highly correlated in the way that uncertain users in one classifier would be very certain in another. E.g, a male that does not write anything gender specific, have very gender specific friends. This is, however, probably not that likely. It is probably more likely that a very masculine profile will have both masculine friends and tweet more commonly about masculine subjects. Still, some of the users might provide some useful information through the combinations.

It was previously mentioned that there are some limitations on the size of our datasets. Particularly, the friends data is quite small. Some images are also missing, making the intersection between friends and images even smaller. When combining classifiers, some methods require a second training set. So while there might be enough data to test the performance of a single classifier, the combined classifier can suffer due to the limited size. It is possible to retrieve more friends data, and it is probably possible to get more images as well. Therefore, when combining the classifiers under these limitations, we are, in some way, moving away from real world application of the problem. However, from an academic point of view, the combination with limited data is very interesting. Also it gives an estimate of what can be expected with more data.

7.1 Methods for combining classifiers

When data from different sources is available, it might be better to combine separate classifiers, as opposed to training one large classifier on all features. This is particularly beneficial when the features are very different (e.g. images and words). Specialized classifiers can be fit to the different features and later combined in a smart way.

There are many ways of combining classifiers suggested in the literature. Kuncheva et al. [2001] gives an overview over some of the methods. Xu et al. [1992] divide the problem of combining classifiers into three categories: combining unordered class labels from the classifiers, combining ranked class labels in order of likelihood, and combining real valued outputs from the classifiers (e.g. posterior probabilities).

Possibly, the most intuitive combining rule is through a majority vote. This is a methods that has been shown to give good results, e.g. in the bagging classifier [Breiman, 1996a] and the random forests classifier [Breiman, 2001]. However, according to Hastie et al. [2009], averaging of posterior probability estimates often produce classifiers with lower variance.

In this project the focus was on combinations through the posterior probability estimates. This was done in the form of some fixed combination rules, and through a separate classifiers.

7.1.1 Fixed combination rules

Each user in our datasets can be represented through a set of feature vectors $\mathbf{x}^1, \ldots, \mathbf{x}^R$. Here, $\mathbf{x}^k$ represents a set of features used by a classifier, e.g. $\mathbf{x}^1$ is an image and $\mathbf{x}^2$ a list of friends. The objective is to combine the classifiers to estimate the posterior probabilities $P(j \mid \mathbf{x}^1, \ldots, \mathbf{x}^R)$, where j denotes the class.

Assume the different classifiers give class probabilities as output. These estimates can then be considered an approximation of the true posterior probabilities,

$$f_j^k(\mathbf{x}^k) = P(j \mid \mathbf{x}^k) + \varepsilon_j^k(\mathbf{x}^k). \tag{7.1}$$

Here $f_j^k(\mathbf{x}^k)$ is the estimate of the posterior probability for class j by classifier k on $\mathbf{x}^k$, $P(j \mid \mathbf{x}^k)$ is the true posterior probability, and $\varepsilon_j^k(\mathbf{x}^k)$ is the error made by the classifier.

Let us consider two extreme scenarios: that the features $\mathbf{x}^1, \ldots, \mathbf{x}^R$ are all equal, and that the features are class conditionally independent. For the first case, the different classifiers are fitted to the same data. Thus, the combined posterior probabilities reduce to,

$$P(j \mid \mathbf{x}^1, \ldots, \mathbf{x}^R) = P(j \mid \mathbf{x}^k), \quad \forall k \in \{1, \ldots, R\}. \tag{7.2}$$

If we assume that the estimated probabilities in (7.1) are unbiased, i.e. $\varepsilon_j^k(\mathbf{x^k})$ has expected value of zero, then the error can be reduced through averaging the estimates. This gives the *average combination rule*,

$$f_j(\mathbf{x}^1, \ldots, \mathbf{x}^R) = \frac{1}{R} \sum_{k=1}^{R} f_j^k(\mathbf{x}^k). \tag{7.3}$$

In the second case the features are class conditionally independent, i.e.

$P(\mathbf{x}^1, \dots, \mathbf{x}^R \mid j) = P(\mathbf{x}^1 \mid j) \cdot \dots \cdot P(\mathbf{x}^R \mid j)$. This gives,

$$\begin{aligned} P(j \mid \mathbf{x}^1, \dots, \mathbf{x}^R) &= \frac{P(\mathbf{x}^1, \dots, \mathbf{x}^R \mid j) P(j)}{\sum_{j'} P(\mathbf{x}^1, \dots, \mathbf{x}^R \mid j') P(j')} \\ &= \frac{\frac{1}{P(j)^{R-1}} \prod_k^R P(j \mid \mathbf{x}^k)}{\sum_{j'} \frac{1}{P(j')^{R-1}} \prod_k^R P(j' \mid \mathbf{x}^k)}. \end{aligned} \tag{7.4}$$

Here $P(j)$ is the prior probability for class j, and can be estimated as the proportion of males and females in the training set. Let $\hat{\pi}_j$ denote this estimate. The estimated posterior probabilities can then be calculated through the *product combination rule* (or multiplication rule),

$$f_j(\mathbf{x}^1, \dots, \mathbf{x}^R) = \frac{\frac{1}{\hat{\pi}_j^{R-1}} \prod_k^R f_j^k(\mathbf{x}^k)}{\sum_{j'} \frac{1}{\hat{\pi}_{j'}^{R-1}} \prod_k^R f_{j'}^k(\mathbf{x}^k)}. \tag{7.5}$$

The two methods presented show how independent and very highly correlated classifiers can be combined. In many cases, however, the features are somewhere in between, and it is not certain which methods is best. As the methods are simple to apply to the problem, it is common to just test both.

Tax et al. [2000] show that for two classes, under some reasonable conditions and equal class probabilities, the product combination rule is just a rescaled version of the average combination rule. Thus, in many cases their performance can be quite similar.

Other similar rules for combining probabilities include the minimum rule, maximum rule, and median rule, see Kittler et al. [1998]. Of these, only the maximum rule was applied in this thesis. Simply stated, the max rule follows the decision of the classifier that is the most certain (highest probability estimate for a data point). If the estimates are good, then this method makes a lot of sense. It is analogous to trusting someone claiming to be an expert.

Kittler et al. [1998] also show that combining classifiers through summation of the posterior probabilities is less sensitive to estimation errors than the other schemes. Thus giving a more robust combination rule.

7.1.2 Combination through a classifier

The rule-based methods above are very dependent on good probability estimates, and are therefore vulnerable to errors in the base classifiers. According to Duin [2002], rule-based combination methods are almost always suboptimal. Data-driven approaches, on the other hand, use data to compensate for errors in the base classifiers. Typically, a new classifier is given the outputs from the base classifiers, and is then trained, assigning weights to the different inputs.

Another benefit is that the inputs to the combination classifier do not need to be of the same scale. A base classifier that does not provide posterior probabilities can still be used as input. For rule-based methods, a probability mapping has to be created instead.

When a classifier is trained on the output of other classifiers, some considerations about the training set need to be addressed. Duin [2002] discuss two different scenarios. The first is to train the base classifiers on a training set, and then train the combination classifier on a different set. This is the preferred method, as possible overfitting in the base classifier will, to some extent, be corrected for in the combination classifier. On the down side, this requires two training sets. If there is limited data, the sets might be too small for any real benefit of the combination.

The second scenario is to train the combination classifier on the same set as the base classifiers. While this does not require a larger dataset, it can easily result in an overly confident classifier. Overfitting in the base classifiers will not be corrected for, but can rather be reinforced. Duin [2002] recommend training weak base classifiers to alleviate this problem.

For more rigorous investigation on the topic of training a classifier on the output from other classifiers, see e.g. Wolpert [1992].

7.2 Missing data

As mentioned in the introduction of this chapter, there are some problems with missing data. Some pictures are missing (missing around 15%), and the friends data might not be large enough. This is handled in a couple of ways. As only images are considered missing, they can be handled through the construction of two classifiers, one with images and one without.

Another problem is that some base classifiers require larger training sets than others. Text classifiers, for instance, need more training data than the friends classifier. That was handled by adding additional tweets to the training sets used by the text classifiers. The tweets were added in a way that no user would end up in both training and test sets. However, the proportion of female users might not be the same as the proportion of female posts, changing the prior probabilities of the classifiers. As previously explained, the classifiers reweigh the data, alleviating this problem. Nevertheless, data-driven approaches have additional opportunity to correct for this bias, possibly making them more suited than the rule-based approaches.

7.3 Combining without images

First, the images were excluded altogether. Most of the classifiers discussed were included in the combination classifiers. This include logistic regressions on,

- tf-idf features from friend IDs

- 2 to 5 grams tf-idf features from tweets (text)
- 2 to 5 grams tf-idf features from text in descriptions
- 1 to 3 grams tf-idf features from emojis and emoticons from texts
- 3 to 5 grams tf-idf features from hashtags

Here, the random forests classifier on the handmade information features in Section 3.1 was not included. Originally, it was, but that resulted in worse results than leaving it out. Also, both binaray and tf-idf friends features were tested, but tf-idf seemed to give slightly better results.

As previously mentioned, the intersection between the datasets is quite limited. All the classifiers, except for the friends classifier, require quite large training sets. As stated in Section 7.2, the training sets for these classifiers were augmented with additional tweets, thus getting the same individual performance as previously reported.

A couple of different approaches to the combination of classifiers has been discussed. Here, six of them were tested and compared. The output probabilities from the base classifiers were combined in a rule based matted using averaging, multiplication and the maximum. A logistic regression and a random forests classifier were fitted to the probability estimates and trained on the same set as the base classifiers. And finally, the training set was split into two, and a logistic regression was fitted to the second training set, while the base classifiers were fitted to the first. All the results are displayed in Table 7.1. The reason why the test set of the last logistic classifier is slightly larger than the rest, is a results of a different partitioning scheme. This should, however, not affect the results in any way.

From the tables it looks like all methods perform more or less equally. The thresholds and parameters for all the different methods were obtained through tuning. None of the methods have higher accuracy or f1-score than the friends classifiers, but the results are a bit more balanced.

The multiplication rule requires estimates of the prior probabilities. As all classifiers are reweighed to be balanced, these were set to 0.5. By instead setting the estimates to the class proportions, the same scores were obtained, but with quite horrible posterior probability estimates (table and figure not included).

As there was limited data, the relative sizes of the two training sets used in Table 7.1f needed to be tuned. It was found that approximately 6 000 of the users should be used to train the base classifiers, and the remaining 300 to train the combination classifier. This suggests that as long as the base classifiers give good estimates, the combination classifier does not need much training.

To further investigate the results, the estimated probabilities for the classifiers were plotted in Figure 7.1. It shows that the quality for the probability estimates differ a lot more than the classification scores. Averaging clearly results in a lack of confidence in the estimates, while random forests might do the opposite. The multiplication rule, max rule, and the logistic regressions all have quite decent probability estimates. Interestingly, the histograms (blue lines) of the logistic regressions *(a) Log. reg with same training set as base classifiers.*

	precision	recall	f1-score	prop.
f	0.81	0.83	0.82	0.41
m	0.88	0.87	0.87	0.59
total	0.85	0.85	0.85	3152

(b) R.F. with 500 trees. Same training set as base classifiers.

	precision	recall	f1-score	prop.
f	0.84	0.79	0.81	0.41
m	0.86	0.89	0.88	0.59
total	0.85	0.85	0.85	3152

(c) Multiplication rule.

	precision	recall	f1-score	prop.
f	0.82	0.79	0.8	0.41
m	0.86	0.88	0.87	0.59
total	0.84	0.84	0.84	3152

(d) Averaging rule.

	precision	recall	f1-score	prop.
f	0.82	0.78	0.8	0.41
m	0.86	0.88	0.87	0.59
total	0.84	0.84	0.84	3152

(e) Maximum rule.

	precision	recall	f1-score	prop.
f	0.81	0.8	0.8	0.41
m	0.86	0.87	0.87	0.59
total	0.84	0.84	0.84	3152

(f) Log. reg. with different training set.

	precision	recall	f1-score	prop.
f	0.79	0.83	0.81	0.41
m	0.88	0.85	0.86	0.59
total	0.84	0.84	0.84	3184

Table 7.1: Results from combining classifiers. The estimated probabilities from log. reg. on friends, tweets, descriptions, emojis/emoticons, and hashtags were used as features.

are somewhat different. Training on the same training set as the base classifiers gives more weight to the highest and lowest probabilities. As this classifier is quite prone to overfitting, this is no surprise. However, the probability estimates do not indicate that the classifier is overfitted.

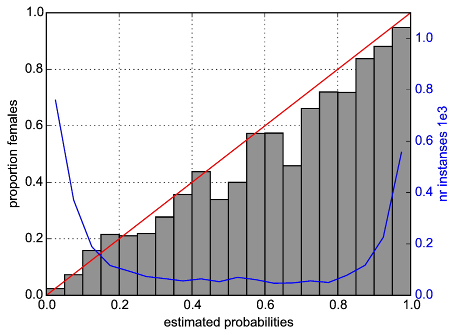
If the probability estimates in Figure 7.1a are compared to those of a logistic regression of just the friends data, see Figure 5.1, the bump in the histogram around the center is flattened. This suggests that there is some benefit of combining the friends data with the other data. However, no conclusions can be made.

Figure 7.2 shows the coefficients of the top left logistic regression in Table 7.1, i.e. the logistic regression trained on same dataset as the base classifiers. In Appendix A, it is explained how the logistic regression models log odds in a linear matter,

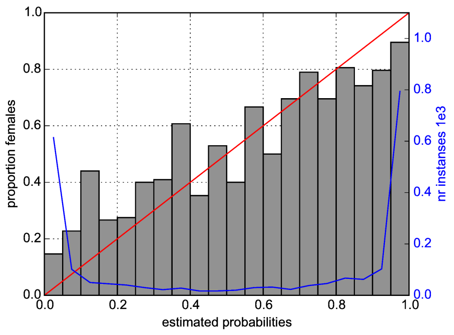
$$\log \frac{P(y=2 \mid \mathbf{x})}{P(y=1 \mid \mathbf{x})} = \beta_0 + \mathbf{x}^T \boldsymbol{\beta}. \tag{7.6}$$

In the combination scheme, $\mathbf{x}$ is the posterior probability estimates from the base classifiers. In Section 4.5 it was argued that the coefficients are not a good measure of the features importance. However, as there are now only five features and they are all on the same scale, the coefficients have some relation to importance. Their interpretation is harder to understand though. This is particularly because the features are all positive probabilities. If the features had been centered around zero, they would have a much more intuitive relationship to the log odds.

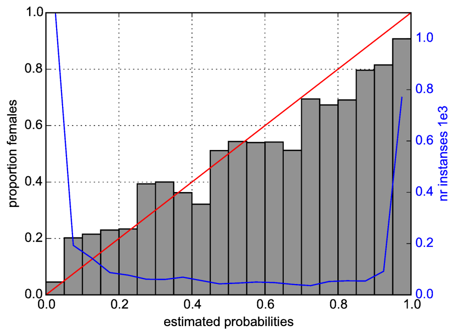
Clearly, the logistic regression puts a lot of weight on the friends, which makes sense. Also the description and text probabilities have decent sized coefficients.



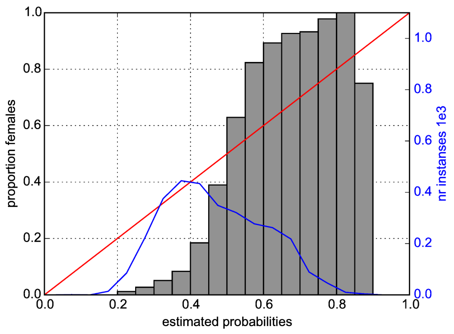
(a) Logistic regression, same training set.



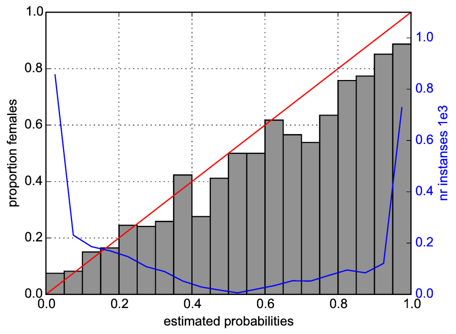
(b) Random forests, same training set.



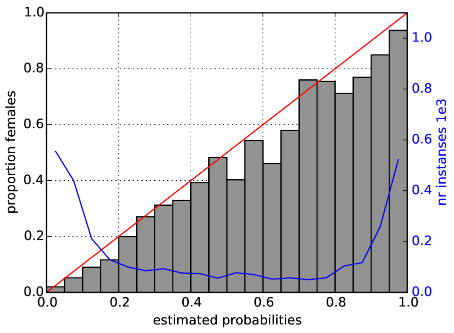
(c) Combining through multiplication.



(d) Combining through averaging.



(e) Combining through max.



(f) Logistic regression, new training set.

Figure 7.1: Estimated probabilities of the combination classifiers in Table 7.1.

Interestingly, both emojis/emoticons and hashtags gave negative coefficients. That means they are both not contributing, but rather just add noise to the model. This is probably because they only have decent predictions for a small subset of the tweets. Take hashtags as an example. Approximately 95% of the tweets do not contain hashtags, so *one* probability is assigned to 95% of the users in the datasets.

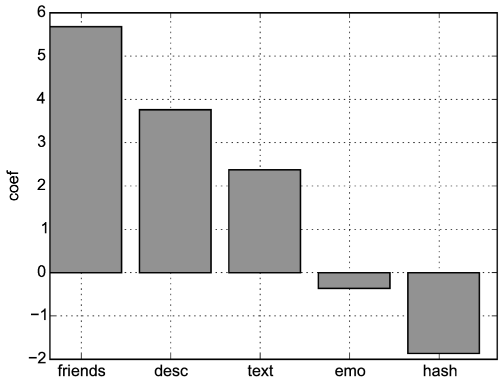


Figure 7.2: Coefficients from logistic regression combination classifier without extra trainingset.

All the experiments above were repeated without emojis/emoticons and hashtags, and there was no apparent difference in the accuracy or the probability estimates (figures not included).

Transformed features

The posterior probability estimates were transformed to log odds, and a logistic regression was fitted to them. This was done through the transformation,

$$(\text{log odds}) = \log \frac{p}{1-p}, \tag{7.7}$$

where p denotes the estimated posterior probability. The motivation behind this is that the logistic regression is linear in its log odds. So, by using the base classifiers' log odds as features, the combined classifier is working in the same scale. If the coefficients could be restricted to sum to 1 (non-negative) and the intercept was removed, then the results would just be a weighted average of the log odds. Note, that this is not the same as a weighted average of the probabilities, as there is a non-linear relationship between the two.

Both the classification performance and the probability estimates of this new logistic regression were indistinguishable from the logistic regression with probabilities as features. Therefore neither tables nor probability plots were included. However, this new method was a lot less sensitive to tuning parameters. As a result, a large variety of coefficients could be used to get the same performance. They did, however, bear resemblance to the coefficients in Figure 7.2.

One possibility why the results were so similar to those of the probability features, could be that (7.7) is quite linear in the range $0.20 < p < 0.80$. However, for large/small p's the function grows quite large/small, giving more weight to these points. Thus, compared to probabilities as features, a classifier with log odds features puts more confidence in base classifiers claiming to be experts.

7.4 Combining with images

The exact same experiments as in Table 7.1 where repeated, but with an additional feature from the image classifier in Section C.2 (logistic regression fitted to GoogLeNet features). The results are also displayed the same way, with classification scores in Table 7.2 and estimated posterior probabilities in Figure 7.3. Note that a slightly smaller test set was used compared to Section 7.3. This is because the intersection between the friends data and image data had to be used. However, none of the training sets were affected by this (except Table 7.2f).

Based on the discussion in Section 7.3, the logistic regression should now be fitted to log odds instead of probability estimates. However, that was found after the experiments in this section were conducted, and because of time constraints the results were note rerun with the transformed features. However, this is not expected to have much impact on the performance.

(a) Log. reg with same training set as base classifiers.

	precision	recall	f1-score	prop.
f	0.84	0.85	0.85	0.38
m	0.91	0.9	0.9	0.62
total	0.88	0.88	0.88	2737

(b) R.F. with 500 trees. Same training set as base classifiers.

	precision	recall	f1-score	prop.
f	0.86	0.8	0.83	0.38
m	0.88	0.92	0.9	0.62
total	0.87	0.87	0.87	2737

(c) Multiplication rule.

	precision	recall	f1-score	prop.
f	0.87	0.84	0.86	0.38
m	0.91	0.92	0.91	0.62
total	0.89	0.89	0.89	2737

(d) Averaging rule.

	precision	recall	f1-score	prop.
f	0.88	0.83	0.86	0.38
m	0.9	0.93	0.91	0.62
total	0.89	0.89	0.89	2737

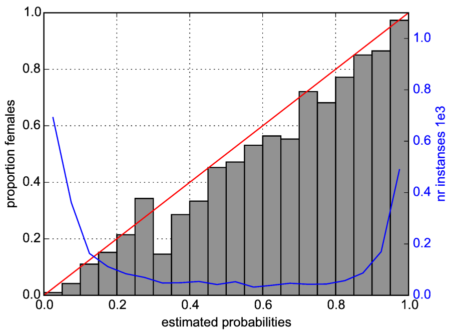
(e) Maximum rule.

	precision	recall	f1-score	prop.
f	0.84	0.84	0.84	0.38
m	0.9	0.9	0.9	0.62
total	0.88	0.88	0.88	2737

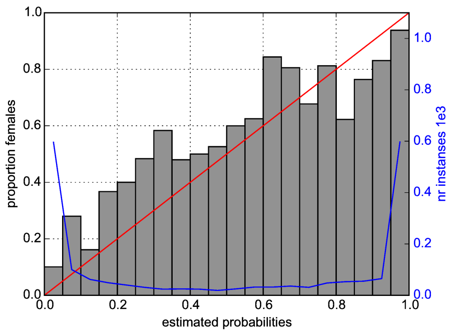
(f) Log. reg. with different training set.

	precision	recall	f1-score	prop.
f	0.85	0.88	0.87	0.38
m	0.93	0.91	0.92	0.62
total	0.9	0.9	0.9	2776

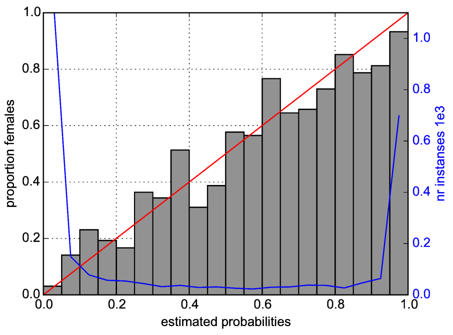
Table 7.2: Results from combining classifiers. The estimated probabilities from log. reg. on images, friends, tweets, descriptions, emojis/emoticons, and hashtags were used as features.



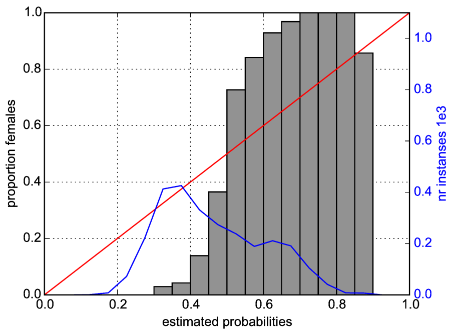
(a) Logistic regression, same training set.



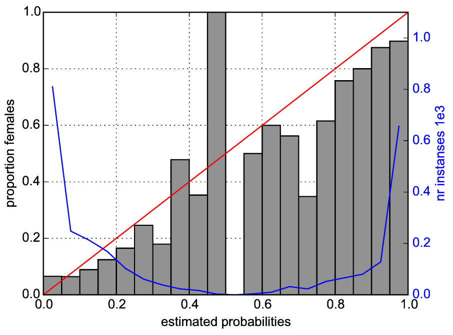
(b) Random forests, same training set.



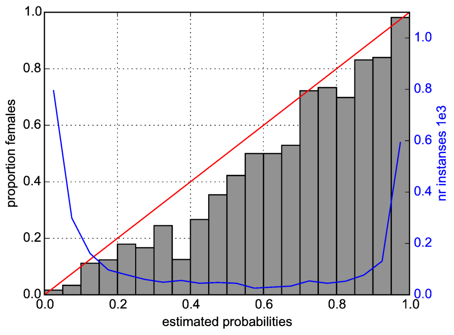
(c) Combining through multiplication.



(d) Combining through averaging.



(e) Combining through max.



(f) Logistic regression, new training set.

Figure 7.3: Estimated probabilities of the combination classifiers in Table 7.2.

Immediately, it is clear that the images has a positive effect on the classification scores. Also here, there is not much difference in accuracy between the different classifiers, though slightly more than previously. The posterior probability estimates are possibly not quite as good as without the image features. The logistic regressions, the multiplication rule, and the max rule still give the best probability estimates.

As before, the size of the logistic regression's second training set was tuned. This time, however, the performance was not as sensitive to the tuning ratio. Everything from equal sized training sets, to the same training sets as for the previous classifier seemed to work fine. Though small differences, we ended up with a base classifier training set of 5 300, and a combination classifier training set of 1 000.

Finally the coefficients of the logistic regression without an additional training set was plotted in Figure 7.4. Also here the emojis/emoticons and hashtags have negative coefficients. Surprisingly, images have approximately the same coefficient as the text. Considering that the image classifier was by far the second best individual classifier, this was somewhat a surprise.

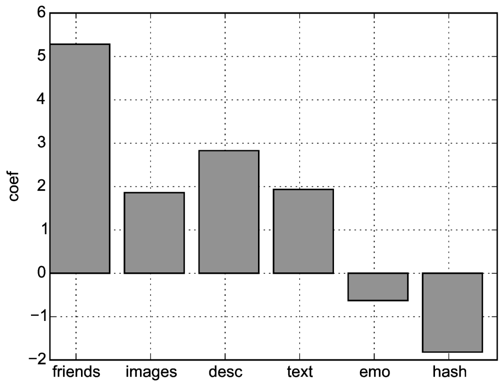


Figure 7.4: Coefficients from logistic regression combination classifier trained on same data as base classifiers. Same as in Table 7.2a.

Chapter 8

Summary

In this thesis, different method for predicting gender of Norwegian Twitter accounts were investigated. The data was mainly collected by Intelligent Communications AS [1]. The main areas investigated were the tweets (text), user descriptions, profile images, and friends of the accounts. These were first analyzed separately, and later combined in an attempt to obtain better classifiers.

For the tweets and descriptions, tf-idf transforms of n-grams (characters) were passed as features to logistic regressions. Both resulted in accuracy around 0.70, though the description classifier was only tested on the subset of accounts that provided descriptions (70%).

Features from the images were obtained through a pre-trained convolutoinal neural net (GoogLeNet), and passed to a logistic regression. This resulted in accuracy of around 0.80 for the subset of accounts providing profile images (85%).

The friends information was also transformed through the tf-idf scheme, and a logistic regression was fitted to it. This resulted in accuracy around 0.85. This was the best individual classifier created.

The best classifiers' posterior probability estimates were combined, yielding a total balanced accuracy around 0.89. Without the image data, the accuracy dropped to 0.85. The combination schemes used were the averaging rule, the multiplication rule, the maximum rule, combination through a logistic regression, and combination through a random forests classifier. They all performed quite equally, though the averaging rule resulted in quite inaccurate probability estimates.

All classifiers were fitted with reweighted data such that the prior class probabilities became equal. The individual classifier were trained on data with different gender proportions (prior probabilities), and the reweighting corrected for this before the classifiers were combined. However, as discussed in Appendix A.2, this also affected the estimated class probabilities. As a result, throughout the thesis, the logistic regressions generally overestimate the probability for females.

[1] Intelligent Communication AS: http://intelcom.no/

The data was labeled though comparing names of the accounts to names in the Norwegian population. Thus, only a subset with regular names was labeled and used for training and testing. This possible bias was not addressed. Also, the method results in the loss of a very informative feature.

8.1 Further work

In Appendix E, multiple tweets of each user were collected, giving larger texts. As a result, the accuracy of the text classifier was increased to 0.75. This new classifier should replace the text classifier in the combination schemes.

Other information could also be aggregated. Meta information collected over many tweets might be informative. Also, as only 5% of the tweets contain hashtags, they are not particularly good features. By aggregating all hashtags used by an account, the subset of users with hashtag data might become sufficiently large to make a positive contribution to the classifiers.

The images seem to have a lot of potential. By investigating features extracted from different layers, better classifiers might be obtained. Alternatively, a ConvNet could be trained, or fine tuned. If there are problems with limited training data, images from other sources could possibly be used. The simples is just profile images of Twitter accounts that are not Norwegian.

The network data from the friends lists gave the best features investigated in this project. Tweets often contain references to other users. This data could be collected as an alternative form of network data.

In this thesis, the data obtained was roughly from the same time period (withing a year), so *when* the tweets were posted, was not considered. For larger datasets, the time is possibly an important feature, though it might just be better to not include old data at all.

Finally, larger datasets could possibly be used as more tweets are collected. This, however, might require some distributed computing.

Bibliography

Sitaram Asur and Bernardo A. Huberman. Predicting the future with social media. *CoRR*, abs/1003.5699, 2010. URL `http://arxiv.org/abs/1003.5699`. (Cited on page 1.)

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2654–2662. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5484-do-deep-nets-really-need-to-be-deep.pdf`. (Cited on page 91.)

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012. (Cited on page 49.)

Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, Jun 2008. ISSN 1077-3142. doi: 10.1016/j.cviu.2007.09.014. URL `http://dx.doi.org/10.1016/j.cviu.2007.09.014`. (Cited on page 43.)

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6287632`. (Cited on page 91.)

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation. (Cited on page 49.)

Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, Mar 2011. ISSN 1877-7503. doi: 10.1016/j.jocs.2010.12.007. URL `http://dx.doi.org/10.1016/j.jocs.2010.12.007`. (Cited on page 1.)

Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. Learning mid-level features for recognition. *2010 IEEE Computer Society Conference on Computer*

Vision and Pattern Recognition, Jun 2010. doi: 10.1109/cvpr.2010.5539963. URL http://dx.doi.org/10.1109/CVPR.2010.5539963. (Cited on page 94.)

G. Bradski. *Dr. Dobb's Journal of Software Tools*, 2000. (Cited on page 46.)

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug 1996a. ISSN 1573-0565. doi: 10.1007/bf00058655. URL http://dx.doi.org/10.1007/bf00058655. (Cited on pages 54, 83, 85, and 86.)

Leo Breiman. Out-of-bag estimation. Technical report, Citeseer, 1996b. (Cited on page 87.)

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. ISSN 0885-6125. doi: 10.1023/a:1010933404324. URL http://dx.doi.org/10.1023/a:1010933404324. (Cited on pages 54, 83, 86, and 87.)

Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification and Regression Trees*. Wadsworth International Group, 1984. (Cited on page 85.)

Alan Bundy and Lincoln Wallen. Difference of Gaussians. *Catalogue of Artificial Intelligence Tools*, page 30–30, 1984. doi: 10.1007/978-3-642-96868-6_57. URL http://dx.doi.org/10.1007/978-3-642-96868-6_57. (Cited on page 43.)

John Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-8(6):679–698, Nov 1986. ISSN 0162-8828. doi: 10.1109/tpami.1986.4767851. URL http://dx.doi.org/10.1109/TPAMI.1986.4767851. (Cited on page 43.)

Chao Chen, Andy Liaw, and Leo Breiman. Using random forest to learn imbalanced data. Technical report, 2004. URL http://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf. (Cited on page 88.)

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning, 2011. URL http://infoscience.epfl.ch/record/192376/files/Collobert_NIPSWORKSHOP_2011.pdf. (Cited on page 49.)

Adele Cutler and Guohua Zhao. Pert-perfect random tree ensembles. *Computing Science and Statistics*, 33:490–497, 2001. (Cited on page 83.)

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005. doi: 10.1109/cvpr.2005.177. URL http://dx.doi.org/10.1109/CVPR.2005.177. (Cited on page 43.)

Thanh-Nghi Do, Philippe Lenca, Stéphane Lallich, and Nguyen-Khang Pham. Classifying very-high-dimensional data with random forests of oblique decision trees. *Studies in Computational Intelligence*, page 39–55, 2010. ISSN 1860-9503. doi: 10.1007/978-3-642-00580-0_3. URL http://dx.doi.org/10.1007/978-3-642-00580-0_3. (Cited on page 27.)

Cıcero Nogueira dos Santos and Maıra Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING), Dublin, Ireland*, 2014. URL http://www.anthology.aclweb.org/C/C14/C14-1008.pdf. (Cited on page 22.)

R.P.W. Duin. The combining classifier: to train or not to train? *Object recognition supported by user interaction for service robots*, 2002. doi: 10.1109/icpr.2002.1048415. URL http://dx.doi.org/10.1109/ICPR.2002.1048415. (Cited on pages 55 and 56.)

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, Aug 1997. ISSN 0022-0000. doi: 10.1006/jcss.1997.1504. URL http://dx.doi.org/10.1006/jcss.1997.1504. (Cited on page 46.)

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Mar 2006. ISSN 1573-0565. doi: 10.1007/s10994-006-6226-1. URL http://dx.doi.org/10.1007/s10994-006-6226-1. (Cited on page 83.)

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010. URL http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_GlorotB10.pdf. (Cited on page 96.)

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011. URL http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2011_GlorotBB11.pdf. (Cited on page 95.)

Robert M Haralick. Ridges and valleys on digital images. *Computer Vision, Graphics, and Image Processing*, 22(1):28–38, Apr 1983. ISSN 0734-189X. doi: 10.1016/0734-189x(83)90094-4. URL http://dx.doi.org/10.1016/0734-189X(83)90094-4. (Cited on page 43.)

Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988. (Cited on page 43.)

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*, volume 2. Springer, 2009. URL http://statweb.stanford.edu/~tibs/ElemStatLearn/. (Cited on pages 45, 54, 80, 81, 86, 87, 90, and 91.)

Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Machine Intell.*, 20(8):832–844, 1998. ISSN 0162-8828. doi: 10.1109/34.709601. URL http://dx.doi.org/10.1109/34.709601. (Cited on page 83.)

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 1991. (Cited on page 91.)

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov 1997. ISSN 1530-888X. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735. (Cited on page 91.)

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. doi: 10.5281/zenodo.15423. (Cited on page 32.)

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe. *Proceedings of the ACM International Conference on Multimedia - MM '14*, 2014. doi: 10.1145/2647868.2654889. URL http://dx.doi.org/10.1145/2647868.2654889. (Cited on pages 49 and 50.)

Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. *CoRR*, abs/1412.1058, 2014. URL http://arxiv.org/abs/1412.1058. (Cited on page 22.)

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL http://www.scipy.org/. [Online; accessed 2015-11-05]. (Cited on page 100.)

G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29(2):119, 1980. ISSN 0035-9254. doi: 10.2307/2986296. URL http://dx.doi.org/10.2307/2986296. (Cited on page 85.)

Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. URL http://arxiv.org/abs/1408.5882. (Cited on page 22.)

Gary King and Langche Zeng. Logistic regression in rare events data. *Political Analysis*, 9(2):137–163, 2001. URL http://pan.oxfordjournals.org/content/9/2/137.abstract. (Cited on page 81.)

J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *IEEE Trans. Pattern Anal. Machine Intell.*, 20(3):226–239, Mar 1998. ISSN 0162-8828. doi: 10.1109/34.667881. URL http://dx.doi.org/10.1109/34.667881. (Cited on page 55.)

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. (Cited on pages 93, 94, 95, and 96.)

Ludmila I. Kuncheva, James C. Bezdek, and Robert P.W. Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern Recognition*, 34

(2):299–314, Feb 2001. ISSN 0031-3203. doi: 10.1016/s0031-3203(99)00223-x. URL http://dx.doi.org/10.1016/S0031-3203(99)00223-X. (Cited on page 53.)

Håvard Kvamme. An investigation into statistical classification using trees. Technical report, NTNU, 2015. URL https://github.com/havakv/Project/blob/master/project.pdf. (Cited on pages 86 and 87.)

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. ISSN 0018-9219. doi: 10.1109/5.726791. URL http://dx.doi.org/10.1109/5.726791. (Cited on pages 43 and 95.)

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. URL http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf. (Cited on pages 89 and 96.)

Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. *2011 International Conference on Computer Vision*, Nov 2011. doi: 10.1109/iccv.2011.6126542. URL http://dx.doi.org/10.1109/ICCV.2011.6126542. (Cited on page 43.)

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml. (Cited on page 23.)

D.G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999. doi: 10.1109/iccv.1999.790410. URL http://dx.doi.org/10.1109/ICCV.1999.790410. (Cited on page 43.)

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. *CoRR*, abs/1412.0035, 2014. URL http://arxiv.org/abs/1412.0035. (Cited on page 94.)

D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society B: Biological Sciences*, 207(1167):187–217, Feb 1980. ISSN 1471-2954. doi: 10.1098/rspb.1980.0020. URL http://dx.doi.org/10.1098/rspb.1980.0020. (Cited on page 43.)

K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Machine Intell.*, 27(10):1615–1630, Oct 2005. ISSN 0162-8828. doi: 10.1109/tpami.2005.188. URL http://dx.doi.org/10.1109/TPAMI.2005.188. (Cited on page 43.)

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a. URL http://arxiv.org/abs/1301.3781. (Cited on pages 22 and 23.)

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013b. URL http://papers.nips.cc/paper/5021-di. (Cited on page 23.)

Alan Mislove, Sune Lehmann, Yong-Yeol Ahn, Jukka-Pekka Onnela, and J Niels Rosenquist. Understanding the demographics of twitter users. *ICWSM*, 11:5th, 2011. URL http://dougleschan.com/the-recruitment-guru/wp-content/uploads/2014/01/Understanding-the-Demographics-of-Twitter-Users-Jukka-Pekka-....pdf. (Cited on pages 1 and 2.)

Brendan O'Connor, Ramnath Balasubramanyan, Bryan R Routledge, and Noah A Smith. From tweets to polls: Linking text sentiment to public opinion time series. *ICWSM*, 11(122-129):1–2, 2010. URL http://www.aaai.org/ocs/index.php/ICWSM/ICWSM10/paper/viewFile/1536/1842/. (Cited on page 1.)

Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Jun 2014. doi: 10.1109/cvpr.2014.222. URL http://dx.doi.org/10.1109/CVPR.2014.222. (Cited on page 97.)

M. Y. Park and T. Hastie. Penalized logistic regression for detecting gene interactions. *Biostatistics*, 9(1):30–50, Apr 2007. ISSN 1468-4357. doi: 10.1093/biostatistics/kxm010. URL http://dx.doi.org/10.1093/biostatistics/kxm010. (Cited on pages 27, 80, and 81.)

F. Pedregosa, 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. *Journal of Machine Learning Research*, 12:2825–2830, 2011. (Cited on pages 99 and 100.)

Stephen M. Pizer, Sarang Joshi, P. Thomas Fletcher, Martin Styner, Gregg Tracton, and James Z. Chen. Segmentation of single-figure objects by deformable m-reps. *Lecture Notes in Computer Science*, page 862–871, 2001. ISSN 0302-9743. doi: 10.1007/3-540-45468-3_103. URL http://dx.doi.org/10.1007/3-540-45468-3_103. (Cited on page 43.)

David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011. URL http://dspace2.flinders.edu.au/xmlui/handle/2328/27165. (Cited on page 10.)

J Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993. (Cited on page 85.)

J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. ISSN 0885-6125. doi: 10.1023/a:1022643204877. URL http://dx.doi.org/10.1023/A:1022643204877. (Cited on page 85.)

Anand Rajaraman and Jeffrey David Ullman. Data mining. *Mining of Massive Datasets*, page 1–17, 2009. doi: 10.1017/cbo9781139058452.002. URL http://i.stanford.edu/~ullman/mmds/ch1.pdf. (Cited on pages 24 and 26.)

J.J. Rodriguez, L.I. Kuncheva, and C.J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1619–1630, Oct 2006. ISSN 2160-9292. doi: 10.1109/tpami.2006.211. URL http://dx.doi.org/10.1109/tpami.2006.211. (Cited on page 83.)

Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Lecture Notes in Computer Science*, page 430–443, 2006. ISSN 1611-3349. doi: 10.1007/11744023_34. URL http://dx.doi.org/10.1007/11744023_34. (Cited on page 43.)

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, pages 1–42, April 2015. doi: 10.1007/s11263-015-0816-y. (Cited on pages 43 and 96.)

Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1983. (Cited on page 99.)

Mark R Segal. Machine learning benchmarks and random forest regression. *Center for Bioinformatics & Molecular Biostatistics*, 2004. (Cited on page 85.)

Aliaksei Severyn and Alessandro Moschitti. Twitter sentiment analysis with deep convolutional neural networks. *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '15*, 2015. doi: 10.1145/2766462.2767830. URL http://dx.doi.org/10.1145/2766462.2767830. (Cited on pages 22 and 23.)

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://www.robots.ox.ac.uk/~vgg/research/very_deep/. (Cited on pages 43 and 96.)

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013. URL http://arxiv.org/abs/1312.6034. (Cited on page 94.)

Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. *Presented at a talk at the Stanford Artificial Project*, 1968. (Cited on page 43.)

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL http://arxiv.org/abs/1412.6806. (Cited on page 94.)

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313. (Cited on page 96.)

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL http://arxiv.org/abs/1409.4842. (Cited on pages 43, 49, and 96.)

David M.J. Tax, Martijn van Breukelen, Robert P.W. Duin, and Josef Kittler. Combining multiple classifiers by averaging or by multiplying? *Pattern Recognition*, 33(9):1475–1485, Sep 2000. ISSN 0031-3203. doi: 10.1016/s0031-3203(99)00138-7. URL http://dx.doi.org/10.1016/S0031-3203(99)00138-7. (Cited on page 55.)

Andranik Tumasjan, Timm Oliver Sprenger, Philipp G Sandner, and Isabell M Welpe. Predicting elections with twitter: What 140 characters reveal about political sentiment. *ICWSM*, 10:178–185, 2010. URL http://www.aaai.org/ocs/index.php/ICWSM/ICWSM10/paper/viewFile/1441/1852Predicting. (Cited on page 1.)

P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001. doi: 10.1109/cvpr.2001.990517. URL http://dx.doi.org/10.1109/CVPR.2001.990517. (Cited on page 46.)

David H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, Jan 1992. ISSN 0893-6080. doi: 10.1016/s0893-6080(05)80023-1. URL http://dx.doi.org/10.1016/S0893-6080(05)80023-1. (Cited on page 56.)

Baoxun Xu, Joshua Zhexue Huang, Graham Williams, Qiang Wang, and Yunming Ye. Classifying very high-dimensional data with random forests built from small subspaces. *International Journal of Data Warehousing and Mining*, 8(2):44–63, 2012. ISSN 1548-3932. doi: 10.4018/jdwm.2012040103. URL http://dx.doi.org/10.4018/jdwm.2012040103. (Cited on page 27.)

L. Xu, A. Krzyzak, and C.Y. Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Trans. Syst., Man, Cybern.*, 22(3):418–435, 1992. ISSN 0018-9472. doi: 10.1109/21.155943. URL http://dx.doi.org/10.1109/21.155943. (Cited on page 54.)

Bai Xue, Chen Fu, and Zhan Shaobin. A study on sentiment computing and classification of sina weibo with word2vec. *2014 IEEE International Congress on Big Data*, Jun 2014. doi: 10.1109/bigdata.congress.2014.59. URL http://dx.doi.org/10.1109/BigData.Congress.2014.59. (Cited on page 23.)

Meliha Yetisgen-Yildiz and Wanda Pratt. The effect of feature representation on medline document classification. In *AMIA annual symposium proceedings*, volume 2005, page 849. American Medical Informatics Association, 2005. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1560754/. (Cited on page 22.)

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *Lecture Notes in Computer Science*, page 818–833, 2014. ISSN 1611-3349. doi: 10.1007/978-3-319-10590-1_53. URL http://dx.doi.org/10.1007/978-3-319-10590-1_53. (Cited on pages 94 and 95.)

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015. URL http://arxiv.org/abs/1509.01626. (Cited on pages 22, 23, and 100.)

Xue Zhang, Hauke Fuehres, and Peter A. Gloor. Predicting stock market indicators through twitter "i hope it is not as bad as i fear". *Procedia - Social and Behavioral Sciences*, 26:55–62, 2011. ISSN 1877-0428. doi: 10.1016/j.sbspro.2011.10.562. URL http://dx.doi.org/10.1016/j.sbspro.2011.10.562. (Cited on page 1.)

Appendices

Appendix A

Logistic regression

The goal of logistic regression is to model the posterior class probabilities and create a classifier based on them. As the probabilities should sum to one, it is obvious that they can not be linear in $\mathbf{x}$. Instead the log odds are assumed to be linear in $\mathbf{x}$. If $y \in \{1, 2\}$ denotes the two classes, the log odds take the following form,

$$\log \frac{P(y = 2 \mid \mathbf{x})}{P(y = 1 \mid \mathbf{x})} = \beta_0 + \mathbf{x}^T \boldsymbol{\beta}. \tag{A.1}$$

The function,

$$\operatorname{logit}(p_{ki}) = \log \frac{p_{ki}}{1 - p_{ki}}, \tag{A.2}$$

where $p_{ki} = P(y = k \mid \mathbf{x_i})$, is called the *logit function* or *logit link*. By inverting the logit function it becomes clear that the posterior probabilities sum to *one*,

$$p_{2i} = \frac{e^{\beta_0 + \mathbf{x}_i^T \boldsymbol{\beta}}}{1 + e^{\beta_0 + \mathbf{x}_i^T \boldsymbol{\beta}}}, \tag{A.3}$$

$$p_{1i} = \frac{1}{1 + e^{\beta_0 + \mathbf{x}_i^T \boldsymbol{\beta}}}. \tag{A.4}$$

There are other possible choices than the logit function, e.g. the probit function based on Gaussian assumptions. They are however not used in this thesis. The logit link is not necessarily used because it is a good assumption. Often it is used as no better assumptions are made.

To find good values for $(\beta_0, \boldsymbol{\beta})$, it is common to find the *maximum likelihood estimator*, or MLE. Let n_i be the number of training samples in group i. A group in this sense is data points with the same covariates $\mathbf{x}_i = \mathbf{x}_j$. Let Z_i the number of points in group i that has class 2. This means Z_i is binomially distributed,

$$P(Z_i = z_i) = \binom{n_i}{z_i} p_{2i}^{z_i} (1 - p_{2i})^{n_i - z_i}. \tag{A.5}$$

Let the N data points constitute M groups. Then the likelihood and log-likelihood are,

$$L(\beta_0, \boldsymbol{\beta}) = \prod_{i=1}^{M} P(Z_i = z_i). \tag{A.6}$$

$$l(\beta_0, \boldsymbol{\beta}) \propto \sum_{i=1}^{M} z_i \log p_{2i} + (n_i - z_i) \log(1 - p_{2i}). \tag{A.7}$$

There is no analytical solution that maximizes the likelihood, so one has to use a numerical optimization algorithm to find the MLE.

After $(\beta_0, \boldsymbol{\beta})$ is found, one can create a classifier by classify to 2 if (A.1) is positive and to 1 if not. However, this assumes the logit link gives accurate probabilities. Often the choice of logit link is based on lack of a better choice so there is little suggesting the probabilities are particularly accurate. It might therefore be better to classify based on a threshold found by e.g. cross-validation.

A.1 Regularization

Typical problems with the standard logistic regression are multicollinearity and overfitting. This can be handled through subset selection methods. Ideally, all possible combinations of features should be investigated, but this is often too computationally expensive. Therefore, greedy methods, like forward-stepwise selection, backward-stepwise selection and forward-stagewise selection [Hastie et al., 2009], are used instead. However, these methods often exhibit high variance, as features are either included or excluded. Shrinkage methods is another approach to regularization that are more continuous, and thus suffers less from the high variability.

Shrinkage is typically done through penalizing the size of the coefficients in the log-likelihood. For an L2 penalty, the loss function becomes,

$$\text{Loss}(\beta_0, \boldsymbol{\beta}) = -\sum_{i=1}^{M} (z_i \log p_{2i} + (n_i - z_i) \log(1 - p_{2i})) + \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2. \tag{A.8}$$

Her the penalty typically does not include the intercept. λ is a hyper parameter that needs to be tuned. This can for instance be done through cross-validation. Park and Hastie [2007] explains how the use of this quadratic penalty helps against problems like collinearity. This gives easier opportunities to code factors using dummy variables, and include higher order interactions.

While logistic regression in general (not considering computational cost) can be fitted without scaling the features, this is no longer the case when penalizing the coefficients. That is because the coefficients are dependent on the magnitude of the features.

While the L2 norm does regularize, there is no variable selection (unless extremely sparse features, see Park and Hastie [2007]). One solution is to use the penalty in combination with a subset selection scheme. Another is to use an L1 penalty instead of the L2,

$$\text{Loss}(\beta_0, \boldsymbol{\beta}) = -\sum_{i=1}^{M} \left(z_i \log p_{2i} + (n_i - z_i)\log(1 - p_{2i})\right) + \lambda|\boldsymbol{\beta}|. \tag{A.9}$$

This actually cause a combination of shrinkage and subset selection. For an explanation of how the L1 penalty sets coefficients to zero, see e.g. Hastie et al. [2009]. Tuning the parameter λ does in no way ensure the optimal subset of parameters, but the penalty has been known to produce good fits and is quite commonly used.

A final option is to combine the two penalties,

$$\text{Loss}(\beta_0, \boldsymbol{\beta}) = -\sum_{i=1}^{M} \left(z_i \log p_{2i} + (n_i - z_i)\log(1 - p_{2i})\right) + \lambda\left((1-\alpha)\frac{1}{2}\|\boldsymbol{\beta}\|_2^2 + \alpha|\boldsymbol{\beta}|\right). \tag{A.10}$$

This introduce another hyper parameter α that controls the weighting between the two penalties.

Applying an L1 penalty to a loss function is often referred to as *lasso* penalty, L2 as *ridge* penalty, and a combination of the to as *elastic net*. Some will argue that these names are only correct for linear regression, but it has been used in other contexts as well.

A.2 Reweighting

In this thesis the number of samples from each class (male, female) is for the most part not equal. When working with imbalanced data some issues needs to be considered. The simplest approach is to just respect that the prior class probabilities are not equal, and fit the logistic regression as before. However, according to King and Zeng [2001], logistic regression can sharply underestimate the probability of rare events. Another approach, is to reweight the classes. This should be done intelligently through tuning parameters and a well constructed metric measuring what we want to accomplish. Alternatively, a quick and dirty way is to just weight the samples based on their class proportions. Typically each training point is then weighted inversely of the class proportions, e.g. N/N_{class}, s.t. both classes have the same contribution to the loss function in (A.7). Note also that the weighting is equivalent to oversampling according to the inverse class proportions, but it is less computationally expensive.

To get a better understanding of the impact of weighting, the log-odds can be investigated.

$$\log \frac{P(y = 2 \mid \mathbf{x})}{P(y = 1 \mid \mathbf{x})} = \beta_0 + \mathbf{x}^T\boldsymbol{\beta}, \tag{A.11}$$

The log odds shows the relationship between the coefficients and the posterior class probabilities. This can be rewritten as,

$$\log \frac{P(y=2 \mid \mathbf{x})}{P(y=1 \mid \mathbf{x})} = \log \frac{P(\mathbf{x} \mid y=2)P(y=2)}{P(\mathbf{x} \mid y=1)P(y=1)} \tag{A.12}$$

$$= \log \frac{P(\mathbf{x} \mid y=2)}{P(\mathbf{x} \mid y=1)} + \log \frac{P(y=2)}{P(y=1)}. \tag{A.13}$$

If the dataset is balanced, the prior probabilities will be perceived as equal, giving,

$$\beta_0 + \mathbf{x}^T\boldsymbol{\beta} = \log \frac{P(\mathbf{x} \mid y=2)}{P(\mathbf{x} \mid y=1)}. \tag{A.14}$$

The original prior probabilities can be introduced again through,

$$\beta_0 + \mathbf{x}^T\boldsymbol{\beta} + \log \frac{P(y=2)}{P(y=1)} = \log \frac{P(\mathbf{x} \mid y=2)}{P(\mathbf{x} \mid y=1)} + \log \frac{P(y=2)}{P(y=1)}, \tag{A.15}$$

and thus only altering the intercept $\beta_0' = \beta_0 + \log \frac{P(y=2)}{P(y=1)}$. This is equivalent to changing the threshold of the logistic regression. Thus balancing the classes can help probability estimates of the smaller classes, while still keeping the original prior probabilities through changing the threshold of the decision. However, it is important to remember that the outputted class probabilities are from the balanced model. Thus adding bias to the estimates. This can also be corrected for through replacing β_0 by β_0' in (A.3) and (A.4).

In this thesis, there are generally more males than females in the datasets. As the logistic regressions are reweighted, the estimated posterior probabilities are biased. The estimates for females are in general to high. Because of the relationship between the probabilities and the log odds, the difference should be increasingly smaller as the probabilities approach 0 and 1. Investigating the reported estimated probabilities through this thesis, it is clear that they coincide very well with the theory. The bars in the plots, giving the proportions of females (e.g. Figure 4.6), are in general too low compared to the estimated probabilities.

Appendix B

Random forests

The term *random forests* is often used for a collection of classification and regression models, averaging an ensemble of decision trees grown in a randomized way. Some of the most well known methods include *bagging* by Breiman [1996a], *random subspace method* by Ho [1998], *perfect random tree ensembles* by Cutler and Zhao [2001], *extremely randomized trees* by Geurts et al. [2006], *rotation forests* by Rodriguez et al. [2006] and *random forests* by Breiman [2001]. In this thesis *random forests* refers to the method by Breiman [2001] and not the general term.

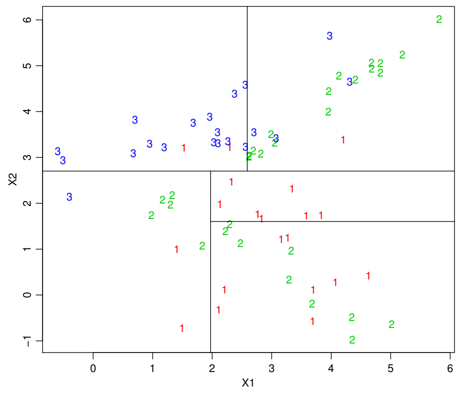
The random forests classifier consists of three parts: decision trees, bootstrap averaging, and randomly choosing subsets of the features. These parts will be introduces through the CART decision trees, bagging, and the full random forests model.

B.1 CART

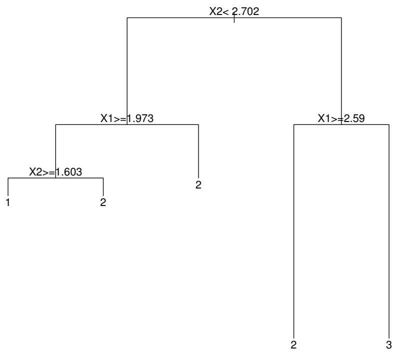
CART stands for classification and regression trees, and is one of the simplest ways to create a decision tree. A classification tree makes local decisions based on a subset of the predictors. This means that the data is split in subsets, that are again split in new subsets (independently of each other) and so on. At the end, the subsets are given a class label based on some vote-measure between the data points, usually the majority. The voting proportions can also be used to estimate the class probabilities, if they are of interest. An example of such a tree can be found in Figure B.1.

It is easy to visualize the splits in a tree. This makes trees highly interpretable. Prediction is done by following the splits down to the decision node, giving the class prediction. An example of such a tree can be found in Figure B.1b.

Unlike linear classifiers like logistic regression, the decision boundaries for trees can take many different forms. Everything from a linear boundary with two end nodes, to a highly complex boundary with the same amount of end nodes as data points, is possible. In this regard, it is important to consider both overfitting and underfitting.



(a) Decisions displayed as boxes.



(b) Decisions displayed as a tree.

Figure B.1: CART run on simulated data in two dimensions.

However, as will be shown later, these issues are not that important when averaged in the random forests classifier, and will therefore not be discussed here.

B.1.1 Building the tree

The CART algorithm split at only *one* variable at a time, i.e. no combinations of features are used in the splits. This means that the domain is split into rectangles, aligned with the axes. Also, each split in CART divide the domain in *two* parts, often referred to as binary splitting. Thus, the splits are done in the simplest way possible.

In Figure B.1 a toy example was simulated to illustrate how the CART algorithm works. To make it easy to visualize, only two features were used. The left figure shows the domains resulting by the splits, while the right a tree view of the splits. Only the tree view generalizes well to higher feature spaces.

The intuition behind CART is quite simple, so next, the splitting decisions is investigated. As it is too computationally expensive to create an optimal tree based on the training data, greedy algorithms for splitting in a sequential matter are used instead. A greedy algorithm makes local optimal choices without much consideration for the global optimum.

A split needs to be based on a criterion and one of the more common (and the one used in this thesis) is the *gini index*. For K classes it is defined as,

$$Q_m(T) = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{B.1}$$

$$\text{where} \quad \hat{p}_{mk} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} I\{y_i = k\}. \tag{B.2}$$

Here T is the tree, m is a node, N_m is the number of data points in node m, y_i is the class of training point i and R_m is the region defined by the node. $\hat{p}_{mk}$ is therefore the proportion of class k in node m. The Gini index increase with the diversity in the node, and gives therefore a measure of *node impurity*. For nodes with only one class it is zero, and for homogeneous nodes (equal amounts of all classes) it gets its maximum value. The greedy step done in CART finds the split that gives the lowest total node impurity, and weight the two nodes by their size.

Now, consider the case where a split is performed on node P. By splitting on the variable x_j, let $R_L(j,s) = \{\mathbf{x}|x_j \leq s\}$ denote the region of the "left" split, and $R_R(j,s) = \{\mathbf{x}|x_j > s\}$ denote the "right". The solution to which variable x_j and split point s that gives the lowest node impurity can then be found by,

$$\{x_j^*, s^*\} = \arg\min_{x_j, s} \left\{ \frac{N_L}{N_P} Q_L(T) + \frac{N_R}{N_P} Q_R(T) \right\}. \tag{B.3}$$

Here x_j and s lies in $\hat{p}_{mk}$ in (B.2). N_L, N_R and N_P are the number of observations in the left, right and parent node respectively.

CART trees are usually regularized by *pruning* a large tree. However, in the random forests algorithm, Segal [2004] found that restricting the size of trees only results in small gains. As pruning is not a common part of random forests it will not be discussed, but the interested reader can find information in Breiman et al. [1984].

There are proposed several other algorithms for building classification trees. Some of the more common methods include ID3 by Quinlan [1986], C4.5 by Quinlan [1993] and its successor See5/C5.0, CHAID by Kass [1980].

B.2 Bagging

Consider a scenario where a learning set $\{\mathbf{x}_i, y_i\}_{i=1}^N$, is given, and a classifier $C(\mathbf{x})$ is created from the set. If a sequence of such sets are given, all drawn from the same distribution, a better prediction could be obtained by aggregating the classifiers $C_b(\mathbf{x})$. A typical way to do this is through the majority vote,

$$C_{agg}(\mathbf{x}) = \text{majority}\, \{C_b(\mathbf{x})\}_{b=1}^B. \tag{B.4}$$

Breiman [1996a] found that this process could be imitated on a single learning set by drawing bootstrap samples from the data. He called this method bagging, or bootstrap aggregating.

Bagging is done by sampling N points (with replacement) from the data set $\{\mathbf{x}_i, y_i\}_{i=1}^N$, and train a classifier $C_b(\mathbf{x})$ on the samples. Repeat this B times and get the aggregated classifier in (B.4).

The use of bootstrap samples makes the computations for each individual classifier independent. This means that the training is easy to parallelize for faster computation. However, if the underlying method is interpretable, this will be lost in Bagging.

It is important to note that even though the votes give proportions of classifiers predicting class k, these proportions should not be used as estimates for the class probabilities. Kvamme [2015] shows this through a simple theoretical experiment. However, often the underlying method used for each single classifier has a probability estimate. An alternative method to (B.4) is to average these probabilities instead,

$$p_{agg,k}(\mathbf{x}) = \text{ave}\left\{p_{b,k}(\mathbf{x})\right\}_{b=1}^{B}, \quad k = 1, \ldots, K, \tag{B.5}$$

$$C_{agg}(\mathbf{x}) = \arg\max_{k} p_{agg,k}(\mathbf{x}). \tag{B.6}$$

This method is more descriptive than (B.4), and has also, according to Hastie et al. [2009], been shown to often produce classifiers with lower variance, especially for small B's. In the context of this thesis, the quality of the posterior probability estimates are very important. Through the thesis, the estimated probabilities from random forests has been shown to often be quite good.

The success of bagging is usually explained through lowering the variance of the classifiers through averaging. Therefore high variance, low biased methods are the best choice of classifiers. Decision trees falls under this category, and bagging is usually associated with tree classifiers.

A more rigorous argument for the success of bagging was made by Breiman [1996a].

B.3 Random forests

Random forests by Breiman [2001] is an extension of the bagging algorithm. By introducing more randomization, it is supposed to further reduce the variance through decorrelating the trees. The method is very similar to bagging. They only differ in how the individual trees are grown. Bootstrap samples are drawn in the same fashion as for bagging. Then, for the creation of each split in a tree, the following steps are done.

1. Select m predictors at random from the total amount of p predictors ($\dim(\mathbf{x}) = p$).
2. Do *one* split, based on these m variables (pick best variable and split-point).

Each tree is grown large and not pruned. Usually the stopping criterion is some minimum terminal node size.

Intuitively, the correlation between the trees should decrease by reducing m, but increase the variance of the individual trees. Therefore m should be considered a tuning parameter. The inventors recommend using $m = \lfloor\sqrt{p}\rfloor$ as a default value for classification.

B.3.1 Why random forests works

In section B.2 an argument was made for why aggregating bootstrap samples can improve the predictions of a classifier. This argument is still valid for the random forests algorithm, but in this section the effect of decorrelating the trees will be investigated.

The simplest, most common argument for the success of random forests is made through the bias-variance tradeoff in the random forests regression. Let $T_i(\mathbf{x})$ denote a trained tree. The random forests prediction for $\mathbf{x}$ is the average prediction over all the B trees,

$$\hat{f}(\mathbf{x}) = \frac{1}{B}\sum_{i=1}^{B} T_i(\mathbf{x}). \tag{B.7}$$

As the trees are created from the same distribution, they are identically distributed. Let σ^2 denote the variance of a tree, and ρ the correlation between trees. It can then be shown (e.g. Kvamme [2015]) that the variance of the prediction is,

$$\mathrm{Var}[\hat{f}(\mathbf{x})] = \rho\sigma^2 + \sigma^2\frac{1-\rho}{B}. \tag{B.8}$$

Large trees are considered relatively unbiased, but with high variance. As the bias is a linear operator, the bias of the ensemble $\hat{f}(\mathbf{x})$ is the same as for an individual tree $T_i(\mathbf{x})$. From (B.8) it is clear that the second term vanish as B grows, thus bagging and random forests manage to reduce the variance of a method without increasing the bias. To further reduce the variance, the first term in (B.8) must be reduced, which can be accomplished by reducing the correlation ρ between the trees.

It is important to note that the error of a classifier can not be decomposed into bias and variance in the same way as for regression. However, classifiers has both systematic error, and error as a result of variability between training sets. So the argument made for regression is somewhat transferable. An more rigorous explanation of how correlation affects the error of a random forests classifier is made by Breiman [2001].

B.3.2 Tuning

One of random forests strengths is that it is relatively easy to tune. Breiman [2001] showed that the algorithm does not overfit as the number of bootstrap samples B increase. So B can either just be set high, or a stopping criterion through a validation set or out-of-bag error [Breiman, 1996b]. Previously it was mentioned that the tree depth could be tuned, but Hastie et al. [2009] argue that fully grown trees usually works just fine and result in one less tuning parameter. The number of splitting variables m, however, is important to consider. As mentioned earlier, $m = \lfloor\sqrt{p}\rfloor$, is a good default value and the optimal m is often close to this value. So tuning random forests can often be do by considering *one* variable, m, that has a good default value. Random forests can therefore be a good choice for a out-of-the-box classifier. This is part of the reason why it was chosen as the non-linear classifier in this thesis.

B.3.3 Reweighting

In Appendix A.2 the logistic regression on imbalanced data was discussed. Random forests classifiers are also affected by imbalance, and this can be handled in similar matters as for the logistic regression. Typically, more weight can be put on the minority class, thus penalizing this class more heavily. This is very similar to reweighting of the logistic regression, and tha same weight, N/N_{class} can be used here. For more on this subject, see e.g. Chen et al. [2004].

Appendix C

Neural networks

Neural networks, or artificial neural networks, are a family of machine learning models loosely inspired by biological neural networks. They can be both supervised and unsupervised, but in this thesis only supervised nets were considered. Also, only classification will be considered, though they can be fitted to regression tasks as well.

Neural networks were quite popular algorithms during 1980's, but later lost some of their domain as other powerful machine learning algorithms were developed. Deeper neural nets had the potential to create higher level interpretations of the data, but was usually very hard to train. In 1989 LeCun et al. [1989] were able to train the first deep net, but it was very computationally expensive, making it somewhat impractical to use. Around 2010 deep neural nets, now often referred to as deep learning, again surfaced, resulting in the development of many new methods. Today deep learning algorithm are considered very powerful and outperforms other machine learning algorithms in several areas. They are nevertheless still computationally expensive, but are enabled through modern distributed computing.

C.1 Perceptrons

The simplest neural nets are probably the single-layer and mulit-layer perceptrons. They are feedforward nets, meaning that the connections between nodes do not form a direct cycle. There seems to be some dispute in the literature whether the single-layer refers to one hidden layer or just an output layer, but here it is referred to as one hidden layer.

The single-layer perceptron is a non-linear, two-stage statistical model used for regression and classification. It consists of an input layer, a hidden layer and an output layer. The structure is typically represented by a network diagram, like in Figure C.1.

Let $\mathbf{x}$ denote a vector representing the input nodes, $\mathbf{z}$ the hidden nodes and $\mathbf{f}$ the output nodes. Then $\mathbf{x}$ is the raw data, and $\mathbf{f}$ is the output probabilities for the

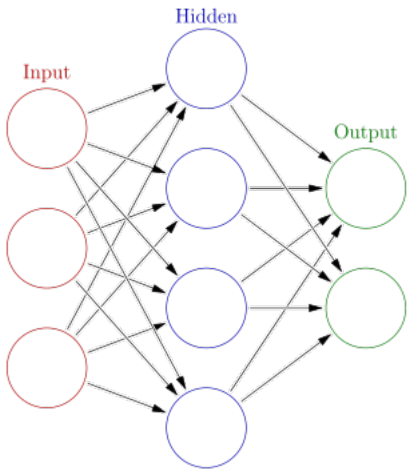


Figure C.1: Example of single-layer perceptron network diagram.

different classes. The M hidden units are calculated from the input,

$$z_m = \sigma(\alpha_{0m} + \boldsymbol{\alpha}_m^T \mathbf{x}), \quad m = 1, \dots, M. \tag{C.1}$$

Here $\sigma(v)$ is usually the *sigmoid* function $\sigma(v) = 1/(1 + \exp(-v)$, and α_{0m}, $\boldsymbol{\alpha}_m$ are parameters, or *weights*, determined through training. The K outputs can then be calculated through,

$$t_k = \beta_{0k} + \boldsymbol{\beta}_k^T \mathbf{z}, \tag{C.2}$$

$$f_k = g_k(\mathbf{t}), \qquad k = 1, \dots, K, \tag{C.3}$$

where $\beta_{0k}, \boldsymbol{\beta}_k$ are weights and $g_k(\mathbf{t})$ is some function. Early work in classification used $g_k(\mathbf{t}) = \mathbf{t}$, but this was later abandoned in favor of the *softmax* function,

$$g_k(\mathbf{t}) = \frac{\exp(t_k)}{\sum_{i=1}^{K} \exp(t_i)}. \tag{C.4}$$

The softmax can be interpreted as estimated posterior probabilities for the K classes.

In Figure C.1 the nodes get input from all the nodes in the previous layer. The structure is said to be *fully connected*. This can result in a high number of parameters, which might be computationally expensive to train. In addition, a fully connected net can be more prone to overfitting [Hastie et al., 2009], so regularization techniques need to be applied.

C.1.1 Fitting perceptrons

In classification tasks, the fit of the net is typically measured through the deviance loss function,

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(\mathbf{x}_i), \tag{C.5}$$

where y_{ik} is the 0/1 coding for the correct class. In this case we have N training points, and K classes. $\boldsymbol{\theta}$ represents all our parameters, and $f_k(\mathbf{x}_i)$ is the posterior probability estimates for the input $\mathbf{x}_i$. If the sigmoid and softmax functions are used, then the network is a linear logistic regression in the hidden units.

The net is trained through gradient decent, called *backpropagation* in this setting. Because of the compositional from of the model, the gradient can be derived using the chain rule for differentiation. Thus the backpropagation can be computed by alternating between forward sweeps through the network, and propagating the error back through the network. The calculations will not be discussed here. More on backpropagation can be found in e.g. Hastie et al. [2009].

C.1.2 Vanishing gradient

A single-layer perceptron can in theory model any structure if the number of hidden units is large enough. However, often adding more layers is an easier approach to learning higher level features [Ba and Caruana, 2014]. So it is attractive to be able to add multiple hidden layers in the model. However, gradients in the backpropagation algorithm are often quite small. Use of the chain rule can result in many small gradients being multiplied together, which causes the gradient to vanish in the lower layers. This was formally identified by Hochreiter [1991]. For more information on the subject see e.g. Hochreiter and Schmidhuber [1997].

Several methods have been suggested to alleviate the problem of vanishing gradients. Typically, the net can be initialized through layer-wise unsupervised pre-training [Bengio et al., 2007]. Deep convolutional neural networks, on the other hand, use rectified linear units.

C.2 Convolutional neural networks

Convolutional neural networks (CNNs or ConvNets) are in many ways quite similar to the neural nets described above. The main difference is that ConvNets where created in an attempt to model the way the brain perceives *images*. Perceptrons are, on the other hand, more general. The focus on images allowed for encoding certain properties into the ConvNets, and vastly reduce the number of parameters in the model.

The nets consists of multiple different layers, each transforming the data through differentiable functions, creating higher level features. There are three main types of layers: *convolution layers*, *pooling layers* and *fully connected layers*.

Parts of this section were based on lecture notes from the CS231n Standford class [1]. Statements from this class are not further referenced.

C.2.1 Convolution layer

The convolution layer is where the algorithm gets its name from, and is the core building block of a ConvNet. Consider a black and white image of size [32x32]. The arguments that are made are easily generalizes to color images [32x32x3], but we use black and white for simplicity. The convolution layer consists of many *filters*, typically of size [3x3] or [5x5]. These filters are matrices of the respective sizes, consisting of the parameters in this layer. Consider one such [3x3] filter. The filter is applied to the image by holding it over a region of the image, e.g. top left [3x3], and doing a inner product between that part of the image and the filter (both flattened). Let $x_{i,j}$ $i, j \in \{1, \dots, 32\}$ denote the pixels in the picture and $w_{i,j}$ $i, j \in \{1, 2, 3\}$ the elements in the filter. The convolutions $z_{k,l}$ can now be calculated by,

$$z_{k,l} = \sum_{i=1}^{3} \sum_{j=1}^{3} x_{i+k-1,j+l-1} w_{i,j}. \tag{C.6}$$

Figure C.2 illustrates this graphically. The red numbers in the yellow filter are the filter's parameters, and the yellow area is where the filter is applied. In this case, the convolution gives out 4, as seen by the lower right element of the pink convolved feature matrix. In the figure, the filter has been applied to the whole image, and all the convolved features are calculated.

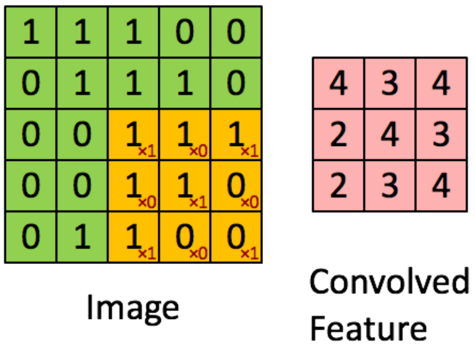


Figure C.2: Example of convolution with one filter. The image is [5x5] and the filter is [3x3].

The intuition behind the convolution layer is the following: If an image patch is highly correlated with the filter, the convolved feature will be high valued. Thus the filter reacts to regions of the image that it is similar to.

[1] CS231n: `http://cs231n.stanford.edu`

The application of the same filter to the whole image is referred to as *parameter sharing*. This parameter sharing, and the local connectivity between an image patch and a convolved feature, vastly reduce the number of parameters compared to a fully connected neural net. In our example with a [32x32] image and a [3x3] filter we only have $3 \cdot 3 = 9$ parameters. A fully connected neural net without parameter sharing would result in $32 \cdot 32 = 1024$ parameters per convolved feature. So if we assume the same dimensionality of the convolved features as the input, this would result in $32^4 = 1\,048\,576$ parameters. Biases have been excluded for simplification. It should, however, be mentioned that every set of parameters includes a bias parameter.

If x and w in (C.6) are flattened locally (part of image interacting with the filter) to x' and w', where $x', w' \in \mathbb{R}^9$, (C.6) can be written as,

$$z_{k,l} = \sum_{i=1}^{9} x'_i w'_i. \tag{C.7}$$

Let now $\bar{w}$ be the reverse of w', and we get,

$$z_{k,l} = \sum_{i=1}^{9} x'_i \bar{w}_{9-i+1}. \tag{C.8}$$

This is a convolution, and as this is equivalent to (C.6), it explains where the layer got its name from.

One layer consists of many filters, creating a matrix of convolved features per filter. Figure C.3 shows an example of one such architecture, and illustrates how the convolution layers works. In the figure there are two convolution layers. The

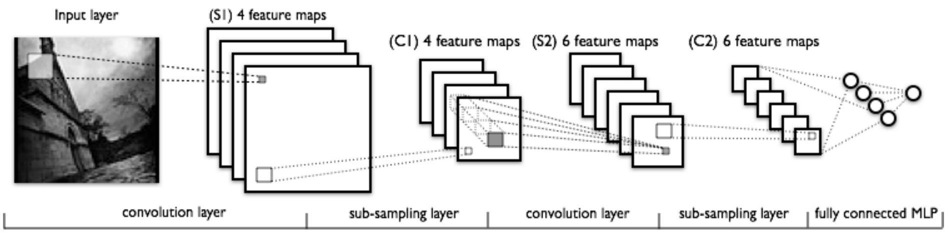


Figure C.3: Example of ConvNet architecture. The sub-sampling layers are pooling layers. The top layer is a multilayer perceptron with one hidden layer.

first used on the actual picture is somewhat intuitive, but the second is harder to interpret. Note that the filters applied to (C1) are three dimensional. The depth of the filter is the same as the depth of the convolved features, i.e. the depth of the filter is the same as number of filters in the previous convolution layer.

Figure C.4 shows the weights from the first layer in the ConvNet by Krizhevsky et al. [2012]. It illustrates how the first layer looks for edges in the image. The next

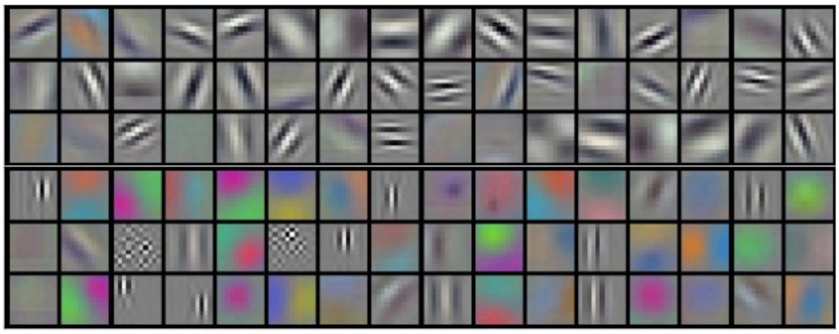


Figure C.4: Weights from first convolution layer in Imagenet by Krizhevsky et al. [2012]. Each weight is [11x11x3] learned from input images of size [224x224x3].

layers, both convolutions and others, are much harder to visualize as the filters are applied to the output from other filters. Some attempts has been made, e.g. Zeiler and Fergus [2014], Mahendran and Vedaldi [2014] and Simonyan et al. [2013], but this will not be covered in this thesis.

C.2.2 Pooling layer

The pooling layer, or sub-sampling layer, is used for two reasons: reducing the dimensionality of the convolved features, and conferring a small degree of translation invariance into the model. The standard approach is through *spatial pooling* [Boureau et al., 2010]. In spatial pooling, a pooling function is applied to a patch of the convolved features. Usually the function is the max value, but average and the L2 norm can also be used. If applied to non-overlapping patches, the dimensionality is reduced. As small changes in the placement of the patch will result in roughly the same output, the pooling provides some translation invariance. The sub-sampling layers in Figure C.3 shows how the patches work graphically.

Typically a patch is of size [2x2] with strides of 2 (jumps two indexes so the patches are disjunct and collectively exhaustive), or size [3x3] with strides of 2 (overlapping). Larger patches result in more translation invariance, but come at the cost of higher information loss. Therefore, larger patches are usually only used in the first layers.

The pooling layers do not have any parameters, so they do not need to be trained. However, the hyper parameters for the size and stride of the patches need to be considered.

Springenberg et al. [2014] suggest to remove the pooling layers and just use the convolution layers. To get dimensionality reduction they suggest using larger strides in some layers.

C.2.3 Fully connected layer

One or more fully connected layers are often used as the last layers in the net. They perform the actual classification. It is also possible to use other classifiers, but a fully connected neural network is very similar to the rest of the framework, and is therefore quite common. Multi-layer perceptrons can be used, but as discussed in Appendix C.1, they can not be particularly deep. This should, however, not be necessary as the previous layers should have created high level features.

C.2.4 ReLU layer

Rectified linear units, or ReLUs, are often not considered layers, but rather part of other layers. However, they play a very important role in the ConvNets, and will be discussed as layers here. ReLU is the non-linear activation function,

$$f(x) = \max(0, x), \tag{C.9}$$

and a linear combination of such functions can be used to approximate arbitrary functions. This is often the only non-linear function in the convolution and subsampling layers. The real strength, however, is that the use of ReLU (compared to e.g. the sigmoid function) alleviates the problem of vanishing gradients. This is further discussed by Glorot et al. [2011]. It is therefore possible to train deeper structures without any unsupervised pre-training. Krizhevsky et al. [2012] shows how some ConvNets with ReLU requires only a small fraction of the training iterations needed with other activation functions.

C.2.5 Architectures

The success of ConvNets are partly owed to their depth. By stacking layers, the nets are able to learn quite complex features. As seen in Figure C.4, the first layer has some form of edge detection. The next layers combine the results from the edge detection into higher level features, which are again combined into even higher level features. Zeiler and Fergus [2014] show how different neurons in a layer are activated with different parts of an image. High level nodes then react to quite complex parts of the pictures. So while the first layer only detects edges, a higher level layer might be able to detect something as complex as faces.

As previously discussed, the pooling layer reduces the dimensionality of the features, and as seen from Figure C.2, so does the convolution layer. This limits the depth of the network. It is therefore quite common to pad the input with zeros on the borders to control the spatial size of the output volumes. This also helps preserve the information close to the borders, so it does not vanish in the first layers.

When building a ConvNet it is important to consider the ordering of the layers. Originally, combinations that alternated between convolutions and pooling were quite common. Now, nets usually have multiple convolutions per pooling layer. Lecun et al. [1998] had the first successful implementations of convolutional nets with LeNet as the best known. This architecture alternated between convolutions and pooling.

AlexNet by Krizhevsky et al. [2012] popularized ConvNets in computer vision through significantly outperforming the other participants in the ImageNet ILSVRC challenge in 2012 [Russakovsky et al., 2015]. They used a much deeper net than LeCun, and stacked multiple convolutions per pooling.

Two of the best known nets today are the winner and runner-up in the ILSVRC 2014 challenge, GoogLeNet by Szegedy et al. [2014] and VGGNet by Simonyan and Zisserman [2014] respectively. The main contribution of GoogLeNet was to dramatically reduce the number of parameters through an *inception module*. VGGNet showed that the depth of the network is a critical component. The best performing VGGNet was extremely homogeneous with only [3x3] convolutions and [2x2] pooling. It was later fond that VGG outperforms GoogLeNet in multiple transfer learning tasks.

Both GoogLeNet and VGGNet have pre-trained models available for the public to use, but we were only able to find a version of GoogLeNet that could be used commercially.

C.2.6 Training

Training a deep neural net often require large amounts of data, and can be very computationally expensive. Thus, super computers or GPUs are often necessary for training them. As a consequence of this, some different approaches to training have been suggested. Typically, one can either train the whole model from scratch, use another trained model as initialization, or just use a trained model as a fixed feature extractor.

Training the full net

Training a full ConvNet can often be very computationally complex compared to other machine learning methods. Procedures usually follow backpropagation as in LeCun et al. [1989]. However, the initialization of a deep net is very important. Training initialized with randomized weights might not work, due to instabilities in the gradient. Simonyan and Zisserman [2014] (VGGNet), for instance, circumvented this by first training a net shallow enough to be trained with random initialization. When training deeper architectures, they used this net as initialization for the bottom and top layers. Later, they found that it was possible to initialize the weights without pre-training, using the random initialization procedure of Glorot and Bengio [2010].

As ConvNets have a very large set of parameters they are quite prone to overfitting. Different regularization approaches have been suggested, but non of them will be discussed here. VGGNet used a combination of weight decay and dropout for regularization. More on these methods can be found in e.g. Srivastava et al. [2014].

Transfer learning

If the dataset at hand is not large enough to train a ConvNet from scratch, or the computations are too extensive for the machines available, another approach can be used. As shown in Figure C.4, the first layer reacts to edges in the pictures. These filters are quite general, so filters trained by one classifier can most likely be used in other classification tasks. Higher level features might also be reasonable to use if the objects in the two classifiers are similar. For instance, high level features from a ConvNet trained on cats can probably be useful for classifying dogs.

With this in mind, we can use a trained ConvNet as a *fixed feature extractor* for the images. The images are run through the net, and the output from one of the layers are used as input to some other classifier (e.g. logistic regression). Which layer we get the features from should be based on how similar the classification tasks are. This method is particularly useful if the dataset is very small.

Another common approach is to *fine-tune* the trained net with the new data. This can be done by changing the top layers of a trained net to fit the problem at hand. Then the whole net is trained on the new data (with initialization from the original data). One might want to consider keeping some of the earlier layers fixed. That can help against overfitting as fewer parameters can vary freely.

Fine-tuning might be a good approach if the dataset at hand is reasonable sized, or the computational power is limited. For more on transfer learning see e.g. Oquab et al. [2014].

Appendix D

Features

D.1 Tf-idf

Tf-idf [Salton and McGill, 1983] stands for *term frequency - inverse document fequency*, and is a common method for extracting features from text. It can be explained through a couple of simple steps.

Word counts, is possibly on of the simplest ways of extracting features from text. It is done by creating a matrix with one row for each document and one column for each word. The elements are the word counts. One problem with counting features is that longer documents will have a higher average count than shorter documents. To adjust for this, the counts can be divided by the number of words in the document, thus obtaining *term frequencies*.

Words that occur in a lot of the documents might not be as informative as words that only occur in a subset. The *inverse document frequency*, downscale weights based on how many of the documents contain the word. Let t denote the term (word) and $D = \{d_1, \dots, d_n\}$ the set of documents. Typically, idf is then defined as,

$$\mathrm{idf}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}. \tag{D.1}$$

Here $|D|$ is the number of documents in the dataset, and $|\{d \in D : t \in d\}|$ is the number of documents where the term t occurs. The tf-idf features can then be calculated through,

$$\text{tf-idf}(t, d) = \mathrm{tf}(t, d)\mathrm{idf}(t, D). \tag{D.2}$$

One common alteration of this is,

$$\text{tf-idf}(t, d) = \mathrm{tf}(t, d)(1 + \mathrm{idf}(t, D)). \tag{D.3}$$

The result of this is that terms that occur in all documents are not ignored, but have an idf of 1. This makes sense as some words might be used by everyone, but some might use it a lot more than others. The python library Scikit-learn [Pedregosa et al., 2011], used in this project, has this alteration as default.

Other versions of tf and idf are also used. For instance a sublinear tf scaling $1 + \log(tf)$. However, they were not used in this thesis.

Although tf-idf was explained through words in text, it can of course be useful in other settings. Typically the terms can be multiple words, or a sequence of letters. This is often called *n-grams*. Tf-idf can possibly be useful in any setting where the information is countable.

In an comparison of multiple text classifiers on multiple data sets, Zhang et al. [2015] found that tf-idf feature outperformed counting features in a majority of the test cases. In their experiments, they also found that tf-idf on words or consecutive words outperformed modern convolutional neural nets for data sets with less than 560 000 training points.

D.2 Scaling features

Neither logistic regression (not penalized) nor random forests are sensitive to scaling, though it might affect the convergence rate of the optimization. However, the coefficients in a logistic regression are dependent on the magnitude of the features. Thus, if it is penalized with the L1 or L2 norm of the coefficients, scaling will affect the results. It is therefore quite common to *standardize* the features, meaning subtracting the sample mean and dividing by the sample standard deviation,

$$x_{scale} = \frac{x - \bar{x}}{s}, \tag{D.4}$$

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{D.5}$$

$$s^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \bar{x})^2. \tag{D.6}$$

Frequency features like counting and tf-idf are sometimes very sparse. By working with sparse matrices like scipy's compressed sparse row matrix (csr by Jones et al. [2001–]), classifiers that that can take advantage of the sparse structure will get a significant speedup. Logistic regression and support vector machines are two examples of such classifiers, while random forests is not. As centering would not preserve this property, one might not want to standardize the features.

Scikit-learn by Pedregosa et al. [2011] recommend to scale sparse features by the max absolute value of the feature,

$$x_{scale} = \frac{x}{\max\{\mathbf{x}\}}. \tag{D.7}$$

This ensures that the scaled features will have a max absolute value of 1. Compared to scaling with the standard deviation, this method is in some ways more sensitive to outliers. For a high value in a feature, all other elements in this feature will become very small. On the other hand, when scaling with the standard deviation, the data is not bounded.

Another approach to preprocessing, is to *normalize* the features. While standardizing is typically performed on the columns of our feature matrix, normalization is typically performed on the rows. Normalization works by dividing the row by e.g. the L1 or L2 norm of the row. This is a reasonable approach if the goal for instance is to quantify the similarity between two samples.

When working with tf-idf features, the term frequency is a normalization of the word counts.

Appendix E

Aggregation of text

Tweets can only contain a maximum of 140 characters. The amount of information available through a tweet is therefore quite limited. Alternatively, multiple tweets from the same users can be collected into larger texts. This is not immediately available, and has to be collected by iterating though our database of tweets.

A dataset was therefore collected, containing all tweets from the users in the friends dataset. This was split into a training and test set, with same proportions as the friends data in Chapter 5. Figure E.1 shows a histogram of number of tweets per account in the training set. Both axes are logarithmic (numbers are correct, but contracted logarithmically). Out of the 6 000 users in the training set, approximately 1 000 users only have one tweet. It is also clear that some of the accounts have over 1 000 tweets.

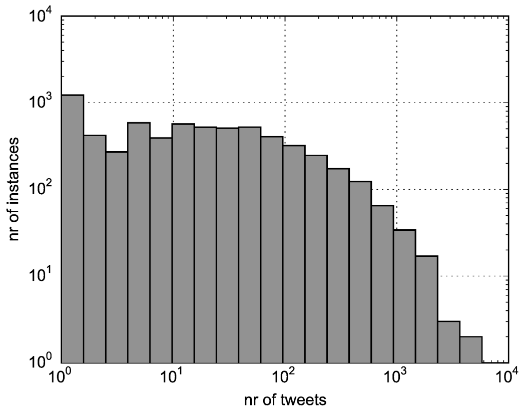


Figure E.1: Histogram showing number of tweets. Note that both axis are logarithmic.

2 to 5 grams tf-idf features were extracted from the data, which resulted in almost 1.5 million features. A logistic regression was fitted to the features, and the classification results are displayed Table E.1. Comparing to the best text classifier on individual

tweets in Table 4.4, the aggregation of tweets is a lot better. Interestingly, also now, the best logistic regression is regularized by the L2 norm, resulting in no subset selection.

The probability estimates were plotted in Figure E.2. Though they are not particularly good, they are probably decent enough. Compared to the single tweet classifier in Figure 4.6, they seem somewhat more noisy, though the trend of overestimating probabilities for females is not apparent.

Table E.1: Classification results from logistic regression on 2 to 5 grams tf-idf features from from text aggregated over tweets.

	precision	recall	f1-score	prop.
f	0.71	0.67	0.69	0.41
m	0.78	0.81	0.79	0.59
total	0.75	0.75	0.75	3152

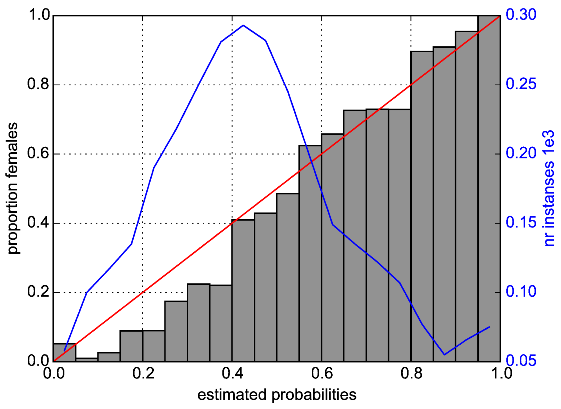


Figure E.2: Probability estimates for logistic regression on 2 to 5 grams tf-idf features from text aggregated over tweets.

This analysis was done somewhat in parallel with the combination of classifiers, and though it probably should have been included, there was not enough time. The analysis presented here is also not that thorough, as is why this was included as an appendix. However, the aggregation of information shows high promise, and would be very interesting to investigate further.

Appendix F

Colors

This appendix is concerned with the color choices available for Twitter accounts. The analysis is not very through, as it was dropped in preference of other, more promising areas. The analysis that was done is included, however, possibly as a warning that colors does not seem the be the best choice of features. Hence, this should not really be considered as part of the thesis, just some additional information for the interested reader.

Every twitter user has the opportunity to configure his or her own profile. This includes some color choices for, among other, text and background. From the collected data, we have access to the five color choices: `profile_background_color`, `profile_link_color`, `profile_sidebar_border_color`, `profile_sidebar_fill_color`, and `profile_text_color`. They all come in the hexadecimal RRGGBB format.

First they were transformed to integers and used as features to a random forests classifiers. The training and test set were the same as in Section 3.1. When mapping a three dimensional color code to a one dimensional feature, colors that are close can be very far from each other, Especially for small changes in RR. Therefore, the random forests classification performance was expected to not be very good. Table F.1 confirms this belief.

Figure F.1 shows bar charts for the proportions of females in the features. The blue lines are histogram of users in thousands. For each chart there is one color that is chosen by more than half of the population. This is probably the default color. Other than that, the figure does not give much information. A couple of the bars are around 0.8, but there are fairly few observations in these. One could try to make classifiers based on only these, but as it is very strange to map colors to a line, this was not pursued further.

The users with default colors were removed from the datasets, and a random forests was fitted to the data. The classification results are displayed in Table F.2. The results are still not particularly good, but there is somewhat more balance between the classes.

Table F.1: R.F. with 200 trees on colors transformed to integers.

	precision	recall	f1-score	prop.
f	0.63	0.3	0.4	0.42
m	0.63	0.87	0.73	0.58
total	0.63	0.63	0.6	11630

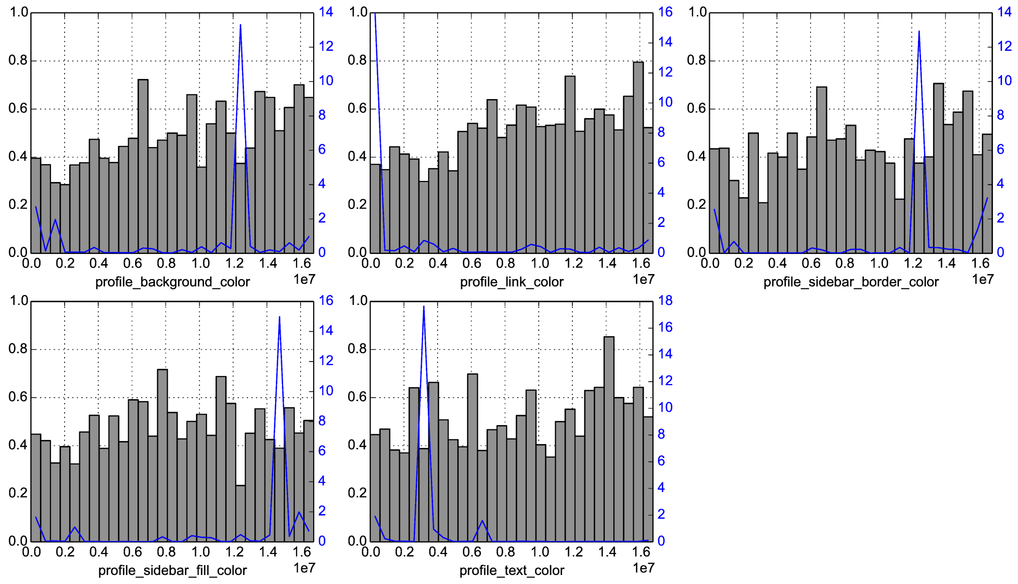


Figure F.1: Bar charts of female proportions in features from training data in Table F.1. The blue lines are histograms of users in thousands.

Next, tree features: RR, GG, and BB, were created from each color. This should make distances between colors a lot more intuitive. Each hexadecimal number is still transformed to an integer. A random forests was fitted to the new features, and from Table F.3, it seems like this new mapping has a positive effect on the classifier. Also here, the female proportions for the individual features were calculated and are displayed in Figure F.2. Each row corresponds to the features: `profile_background_color`, `profile_link_color`, `profile_sidebar_border_color`, `profile_sidebar_fill_color`, and `profile_text_color`. The columns corresponds to RR, GG, and BB. The labels were removed due to space considerations.

Here there are no clear patterns, but there are some high and low proportions that was studied further. A new classifier was based on these most extreme proportions, and only users with these colors were included in the training and test set. There are some tuning parameters that needed to be considered: the with of the bars, the minimum number of users within a bar, and the lower and upper extreme values for the proportions. These were tune, though not particularly thoroughly. The

Table F.2: R.F. with 200 trees on colors transformed to integers. Users with default colors were removed from the dataset.

	precision	recall	f1-score	prop.
f	0.63	0.54	0.58	0.46
m	0.65	0.72	0.68	0.54
total	0.64	0.64	0.64	5733

Table F.3: R.F. with 200 trees on colors split into R G B integer features. User with only default colors were removed.

	precision	recall	f1-score	prop.
f	0.65	0.56	0.6	0.46
m	0.66	0.74	0.7	0.54
total	0.66	0.66	0.65	5733

result of the classifier can be found in Table F.4. From the table it is clear that high accuracy is only obtainable for a very small subset. Also, colors seems to be more descriptive for women than men.

Table F.4: R.F. with 200 trees on colors split into R G B features. Only best separating colors were used.

	precision	recall	f1-score	prop.
f	0.83	0.92	0.87	0.75
m	0.64	0.44	0.52	0.25
total	0.79	0.8	0.79	397

In Figure F.3 the colors of the users included in this training set are displayed. The area of the colors are representative for the number of users. Red and pink seems to be indications of female users, and blue and green might be an indication of a male user.

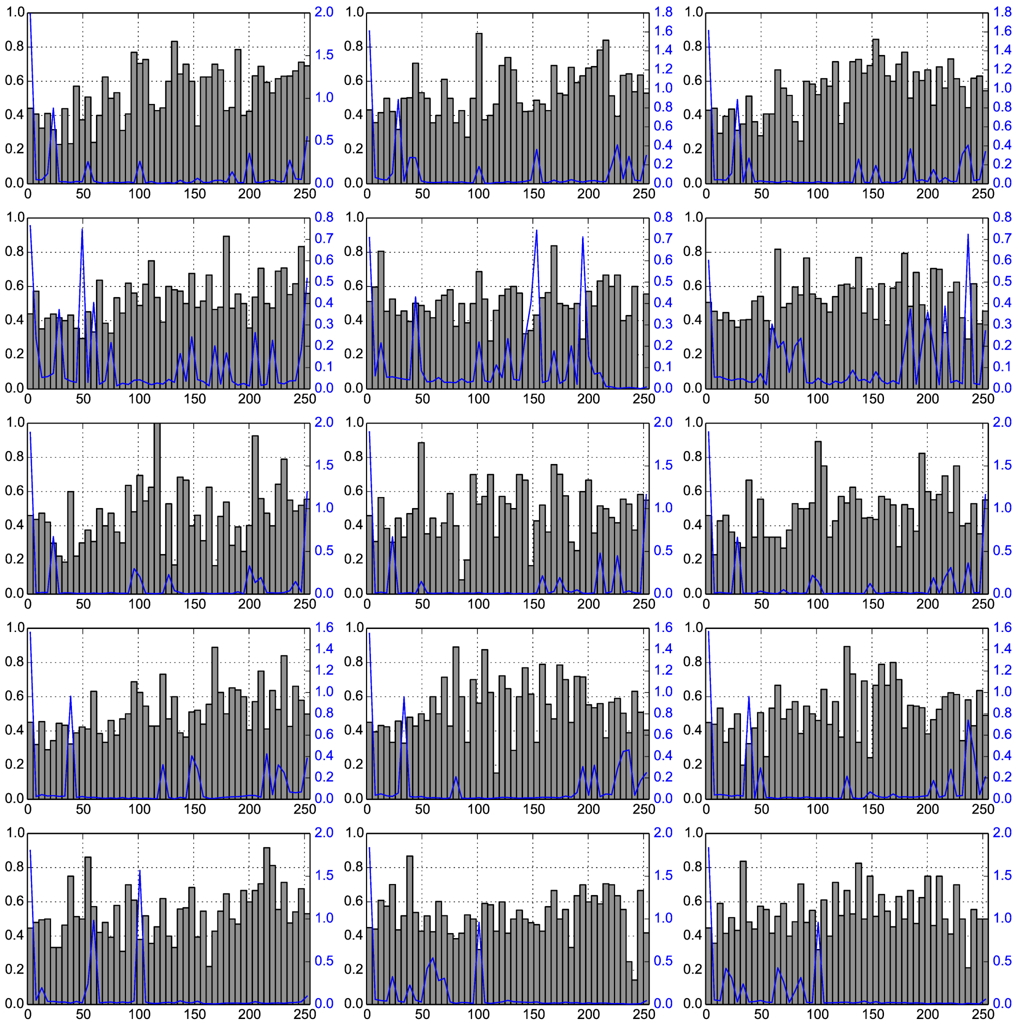
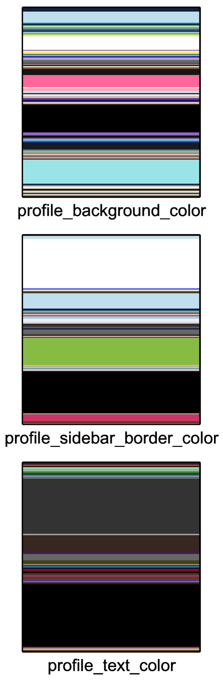
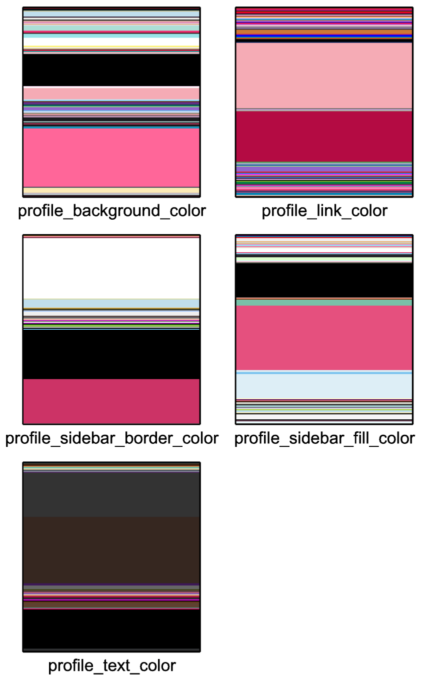


Figure F.2: Female proportions of R G B colors. Each row corresponds to the respective features in Figure F.1, with the same ordering. The columns are RR, GG, and BB. Default colors are removed. The blue lines are users in each bar (in thousands).



(a) Male



(b) Female

Figure F.3: Colors from users in the training set used in Table F.4. The size of the areas corresponds to the number of users.