RADBOUD UNIVERSITY NIJMEGEN

ARTIFICIAL INTELLIGENCE

BACHELOR'S THESIS IN ARTIFICIAL INTELLIGENCE

Identifying Purchase Intentions by Extracting Information from Tweets

Author:
Martijn OELE
S4299019
martijn.oele@student.ru.nl

Internal Supervisor:
Prof. dr. B.E.R. de RUYTER
Faculty of Social Sciences

External Supervisor:
dr. B.J.M. van HALTEREN
Faculty of Arts

February 8, 2017

Abstract

Social Marketing is significantly gaining importance with companies advertising on social media such as Facebook and Twitter. Consumers prefer personalised advertisements that are related to for example their hobbies, work or interests. However, not all companies can afford to spend a lot of money for buying data of their potential customers. Therefore, I want to investigate if an artificial intelligence approach can predict (from existing user created content on social media) if someone is a potential customer for a specific company or product. In my approach I focus on predictions based on streams of short messages only, in contrast to approaches adopted by companies such as for example Facebook that involves many additional sources of data (e.g. mouse trajectories, browse history). The predictions of the artificial intelligence approach are compared to annotations from a human expert. This annotator reads the timeline of all users in the dataset and assigns a label *PC* (Potential Customer) or *non-PC* (no Potential Customer) to the users. There are already many studies that investigate how data models can process natural language in tweets and identify the sentiment of social posts, but combining all these techniques to detect purchase intentions in tweets is a relatively unexplored field.

In my approach, models have been trained using different machine learning algorithms and tested with 10-fold cross-validation. Two experiments have been conducted in which the performance of the these models is evaluated by comparing the model's predictions to the annotations of the human expert. I have trained one model using knowledge-rich features, two models using knowledge-poor features, and a model using both. The results are compared to find out the contribution of the features to each other. The results of the experiments give an answer to the research question: To what extend can AI approach the predictions (potential customer vs. no potential customer) that are assigned by a human expert? The results show that the AI approach classifies a user too often as a potential customer. When the artificial intelligence model uses all information that is available it can identify nearly 90% of the *PC* instances. However, the precision at this threshold is slightly below chance level. There are several reasons for future work. One reason is that the dataset that is used in this study is based on predictions of a human expert and not on actual purchase behaviour. The reliability of the artificial intelligence approach can be improved by using a dataset that contains tweets of users that are actual customers of a company. This dataset can be used to investigate if the artificial model can identify someone's purchase intention before he became a real customer of the company.

Contents

1 Introduction

Since the introduction of Web 2.0, big companies, such as Facebook and Google, know more of us than we can imagine. They do not only store your browse history, but also the time that you spend online, the links you click and the mouse trajectories. Among other things, this data is processed to find out your interests, so you can get personalised advertisements. I am very interested in the role of artificial intelligence in big data. Furthermore, I am also attracted by the strength of marketing on social media. Social Media Marketing is a relatively new approach to Social Marketing. Social Marketing was introduced as a concept in the early 70s. Kotler and Zaltman (1971) used this concept to refer to the promotion of social objectives, such as brotherhood and safe driving. This interpretation of social marketing became less important and when you ask a marketer how they interpret social marketing nowadays, they will explain that social marketing is related to applying marketing through social networks, online or offline. It should be noted that, although my study will focus on purchase intent, that Social marketing is not only related to trying to sell as many products through social networks. In general, Social marketing is used in applying all 4 Ps that are known in the marketing mix: product, price, place and promotion (Thackeray et al., 2008). There exist different purposes of social marketing: viral marketing, increase buyers loyalty, increase product awareness. In this study, I will focus on the latter and I am going to find out if it is possible to efficiently identify purchase intents without involving all the other data sources (e.g. mouse trajectories, browse history) which companies like for example Google and Facebook deploy in their approaches.

It would be too simplistic to assume that big companies only filter on relevant keywords and show advertisements related to the keywords you used in your posts. In their approaches more than keyword filtering is used to ensure that undesirable scenarios are prevented. Such a scenario could for example be that someone receives a promotion for cigarettes after he posted a tweet (a micropost with a maximum of 140 characters on Twitter) about cigarettes in a negative context: "@username: No more cigarettes #diagnosis #lungcancer". Even so, there still exist cases that seem innocent in which advertisements are not filtered, but these cases can nevertheless be very painful for the person itself. A simple example of this would be a woman seeing advertisements of baby clothes 9 months after she started following Facebook pages with information about pregnancy, although she has now stopped searching this because of an unwanted abortion. Such could have been avoided with an approach understanding more of the personal context[1].

To work on this problem, I am going to combine existing techniques (e.g. sentiment analysis[2]) with my own functions to extract features from tweets. Then, I will create a model that tries to predict whether or not someone can be a potential customer. This prediction will only be based on streams of short messages (i.e. the tweet history of a person). I will apply the model to tweets of people that use product-related terms in one of their tweets. The model can be trained using labels that are manually assigned

[1]The approach should thus not only understand the message that is posted, but also why it is posted, the relations between this post and older posts, etc.

[2]Determine whether a message is positive, negative or neutral

by an employee of the marketing division of the company that is used. This employee determines whether the author of tweets is a potential customer (*PC*) or not (*non-PC*) in his opinion. Besides, the annotation, she marks the number of tweets that had to be read before the decision could be made. After a model is trained, it is applied to a subset of the data. How these subsets are formed is explained later. I can evaluate the model by looking at the precision[3] and recall[4]. The evaluation should give an answer to the question: To what extend can AI approach the predictions (potential customer vs. no potential customer) that are assigned by a human expert?

In the next section of this report I discuss previous research that is relevant to this study, what is achieved so far and what I want to improve. In Section 3, I will introduce the approach to create the model and explain how the tweets are collected, processed and annotated. Furthermore, I will explain which features are used and how they are obtained. To conclude, I will compare different machine learning techniques and clarify which one will be uses. The experimental setup and results are explained in section 4. The results are interpreted and related to the literature in section 5. In section 6, dicsusses possible improvements and future work.

2 Related Work

Previous studies have shown that it is possible to apply Natural Language Processing (NLP) and Named Entity Recognition (NER) to tweets (Li et al., 2012) (Liu et al., 2011). However, applying NER to tweets is very difficult because people often use abbreviations or (deliberate) misspelled words and grammatical errors in tweets. Finin et al. (2010) tried to annotate named entities in tweets using crowdsourcing. Other studies used these techniques to apply sentiment analysis to tweets. The first studies used product or movie reviews because these reviews are either positive or negative. Wang et al. (2011) and Anta et al. (2013) analysed the sentiment of tweets filtered on a certain hashtag (keywords or phrases starting with the symbol that denote the main topic of a tweet). These studies merely analyse the sentiment of a tweet about a product after the author has bought it. In this study, I use the Sentiment140 API[5] developed by Go et al. (2009).

2.1 Identifying Purchase Intentions

Kim et al. (2011) investigated purchase intentions for digital items (e.g. online avatars, digital wallpapers, skins) on social networking communities. They concluded that there is a lack of understanding the way in which social network members elect to purchase something. They also mention that is hard to identify on which factors the purchase decision is based. Another study did not only use the sentiment of tweets, but also semantic patterns to analyse purchase intentions on Twitter (Hamroun et al., 2015). Their

[3]Precision: How many selected items are relevant?
[4]Recall: How many relevant items are selected?
[5]http://help.sentiment140.com

approach showed a consistent performance on a Twitter dataset (precision: 55.59%, recall: 55.28%). Gupta et al. (2014) did not use tweets but Questions&Answers sites (Quora[6], Yahoo! Answers[7]) to identify purchase intentions. They used different features extracted from the posts on these websites to assign classes (PI: Purchase Intention and non-PI: no Purchase Intention) to these posts using a Support Vector Machine (SVM). They experimented with different sets of features and used the area under the ROC curve (i.e. a curve of the sensitivity and the aspecificity[8] for a binary classifier) as measurement for the accuracy. The highest AUC (Area Under the Curve) was 0.89 when they used all features together. Precision, recall and AUC are different metrics for the performance of a classification model. The ROC curve is created by plotting the false positive rate against the true positive rate. This curve does not take the precision into account, so one needs all results to compare the AUC from a ROC curve against the precision and recall.

3 Methodology

3.1 Approach

The diagram in figure 1 gives an overview of the approach adopted in this project. Collecting and annotating the data is done at the start of the project and is not repeated. During the annotation process, the moment when a label is assigned is stored. This moment is used later on, so it is passed to the feature extractor together with the true classes (i.e. assigned labels). The steps in the box with denoted with the black dashed line are repeated for each experiment. The steps in the box denoted with the orange dashed line are repeated for each machine learning algorithm. Each combination of an experiment and an algorithm leads to a performance that is shown in the results section.

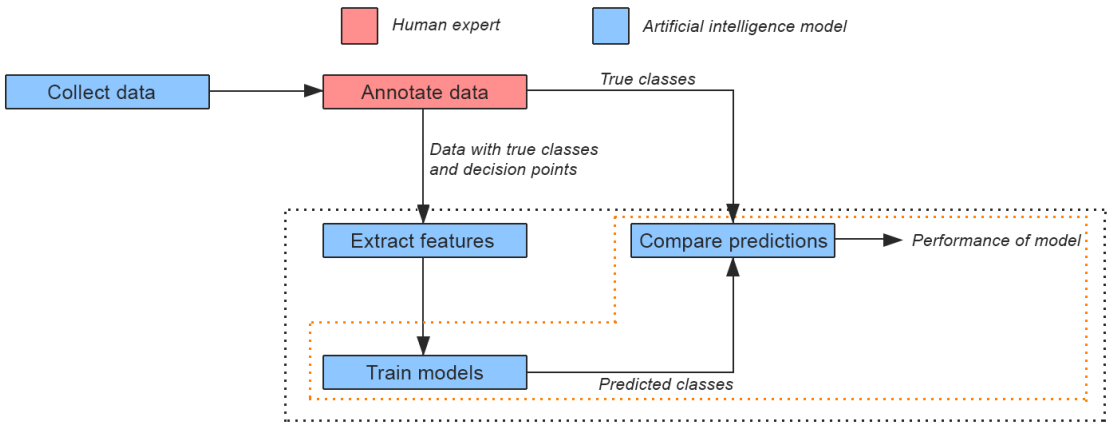


Figure 1: *General Overview of the Approach*

[6]https://www.quora.com
[7]https://www.answers.yahoo.com
[8]1—specificity

3.2 Data Collection and Annotation

3.2.1 Tweets

I have decided to query for tweets about one particular product, so I did not have to experiment with settings of the model that are different for short-term purchases versus long-term purchases. The product that I focussed on in this study is the Philips Hue Lamp. This product is relatively new and expensive, so potential buyers might post their questions and thoughts of the product on Twitter before they potentially buy it. Furthermore, I assume that the majority of the target audience for the Hue Lamp is actively using Twitter.

To collect tweets, I have used 2 separate methods. The first method is the Twitter API. However, due to some limitations of this API (the API uses a rate limit and gives inaccurate query results), I had to find another method. For example, when I query for tweets before a particular date that contain a specific term, it does not return all existing tweets. In other scenarios, it returned an empty set because the Twitter API was not able to search for tweets before a specific date. I wanted to be able to use this 'until date' because I needed Twitter users that wrote several tweets after they used product-related keywords. The Twitter API would return tweets with the product mention from the last couple of days when there was not set an until date. Most of the writers of those tweets did not have enough tweets after the mention, so they were not usable for machine learning. The alternative method I used parses the output of the search page on twitter.com when the input is a query with keywords, an until date and a language. This method receives a list of tweets in chronological order. I have created a list of keywords to find users for the dataset. These keywords are based on the content of the product's website[9] and on specific terms that are used in the posts on the product's Facebook page. The keywords that I have used are *Personal light*, *Personal lamp*, *Ambiance light*, *Automatic light* and the exact combination *Homework ambiance*. *Automatic light* is often used on the Facebook page to promote the new motion sensor. On this page, they also promote the Concentrate Mode of the Philips Hue.

For each user, I downloaded all tweets that were posted in the period between ten days before and two months after the tweet with the keyword. I wanted to use some tweets that were posted before the tweet with the keyword because that particular tweet can be written because the author has bought the related product already and if this is the case, the tweets before that tweet can be relevant. While this would have been interesting, it would have given the human annotator (whose task will be explained later) much more work. Since there is a possibility that the user never wrote about the intention to buy the product, the annotator would annotate this user as a non-potential customer, unless he read everything until the last tweet. Therefore, the annotator would be required to read all tweets of every user and unfortunately this was not possible at this moment. It was hard to decide how the window had to be placed. I had to make a choice between knowing for sure that a person has bought a product and investigating if the user wants to buy more products (but that would

[9] http://www2.meethue.com/en-US

change the research question), or having a human expert that annotates the dataset. The downside to the first scenario is that I do not know if the tweets of the user are useful (i.e. did the user express his purchase intentions). The limitation of the second case are that the dataset cannot be too big, since the human expert cannot spend all his spare time annotating data.

I have used the `sinceID` parameter on queries to download tweets that were posted in the same period as the tweet that contains one of the keywords. For each user, this parameter is set to an `id` that is slightly smaller than the `id` of the tweet with a keyword. In this way, the Java program can collect 40-80 tweets that are posted in a period of 71 days, where the tweet with the keyword is posted at the tenth day. Note that the number of tweets before the tweet with the product-related keyword is different. The size of the timeline should not be expressed as the number of tweets but the number of days, because the number of tweets that people post per day varies. In this way, I ensure that the dataset does not contain users that have only posted 40-80 tweets in more than a year, because it is not likely that a purchase intention can be identified by looking at 40 messages that are posted in one year.

3.2.2 Annotation

I am very grateful that a Global Social Media Analyst at Philips wanted to help with this study. This employee, Ms. Corina Bordeianu, operated as a human expert for this study. Her judgment on people being a potential customer or not functions as ground truth for my model. I created a webpage that reads a JSON file that contains the entire dataset and then displays all tweets in chronological order for every user in the dataset (figure 2). The human expert can read those tweets and click 'positive' or 'negative' at the moment a decision can be made on whether or not the user is a potential customer for the product. The web-interface stores the position when the annotator has made the decision. When all users have been assigned with a class label (1 = potential customer; 0 = non-potential customer) the data is exported to a JSON file.

3.2.3 Privacy

Obviously, only tweets of users that have made their account public are used in this study. According to the Developer Agreement & Policy[10] it is forbidden to display or distribute tweets in a manner that would have the potential to be inconsistent with the Twitter users' reasonable expectations of privacy or to display tweets to any person that may use the data to violate the Universal Declaration of Human Rights. Therefore, I have anonymised the tweets as much as possible, so the author can not be traced via my web-interface. The webpage for annotation can only be reached via an encrypted connection and the annotator should use a a username and password. The webpage only shows the texts of tweets together with the date and time of the tweet. All usernames in tweets are replaced by *@username*. Furthermore, the URLs in tweets are

[10] https://dev.twitter.com/overview/terms/agreement-and-policy

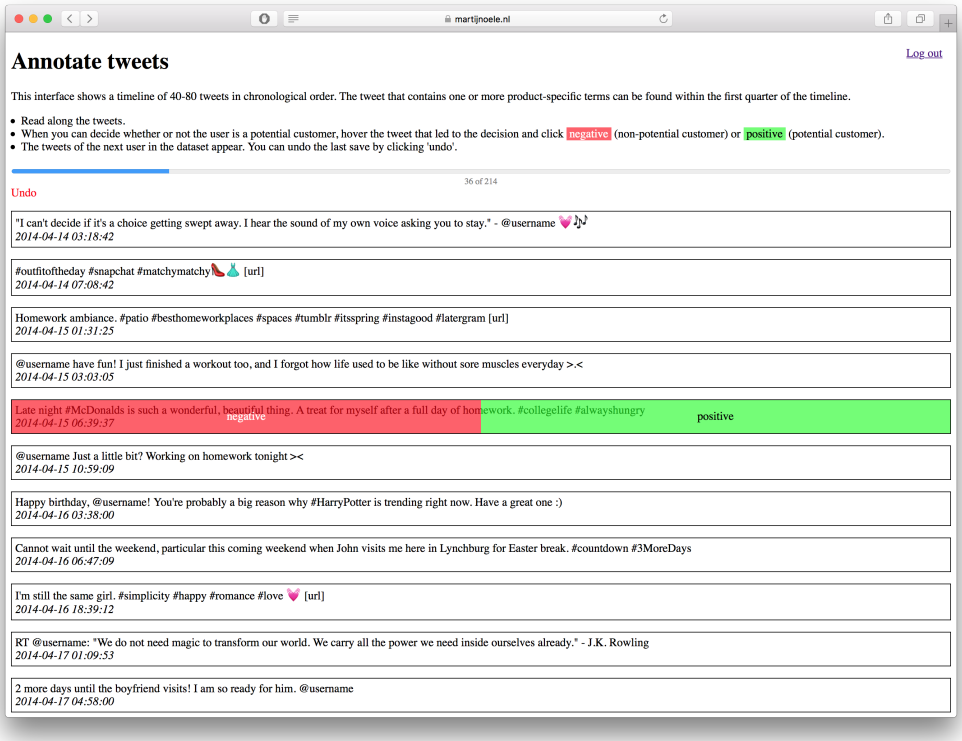


Figure 2: *Web Interface for Data Annotation*

replaced by *[url]* since they were not clickable. The same has been applied to tweets when they are mentioned as examples. The model can read the usernames in tweets in order to recognise possible patterns that may be relevant for machine learning, but the usernames are not stored.

3.3 Java Implementation

To represent the users with their tweets in Java I have created 2 classes. The user is represented by a class `User` which has attributes for the `id`, `username`, `bio` and `timeline`. This `timeline` is filled with a number of `Tweet` objects. Each `Tweet` has an `id`, `text` and `datetime` attribute. Both classes have a method that returns JSON objects. These methods are used when the dataset is exported to JSON to prepare the task of the annotator. When a timeline is used in this report, it refers to the timeline object in the Java implementation, so the end of the timeline means the latest tweet that was downloaded for this user.

Not only collecting tweets but also extracting features from tweets, storing them in temporary files, training the machine learning models and testing those models on unknown test sets of the data happens in Java. Since the features differ for the experiments, you choose the experiment when the features are extracted. Figure 3 shows a couple of screenshots of the Java program.

```
Enter choice: 4
> Training phase
> Read features from file [unigram_features.csv]
> Normalizing data
> Read features from file [skipbigram_features.csv]
> Normalizing data
> Read features from file [abstract_features.csv]
> Normalizing data

------------------------
          MENU
------------------------
 1) Crawl tweets
 2) Extract features experiment 1
 3) Extract features experiment 2
 4) Train & Test experiments
 0) Exit
------------------------
Enter choice: ..*
optimization finished, #iter = 319
nu = 0.6671438119634621
obj = -814.6561414168614, rho = 18.421718130328216
nSV = 121, nBSV = 77
Total nSV = 121
.*
optimization finished, #iter = 231
nu = 0.7269237039070098
obj = -900.4478005047407, rho = 21.75467988812667
nSV = 129, nBSV = 87
Total nSV = 129
..*
optimization finished, #iter = 347
nu = 0.7098008260080115
obj = -866.4020963671097, rho = 41.49785456653764
nSV = 123, nBSV = 85
Total nSV = 123
.*
optimization finished, #iter = 229
nu = 0.7703064342982935
obj = -905.6883268877585, rho = 73.87060930058496
nSV = 127, nBSV = 94
Total nSV = 127
..*
optimization finished, #iter = 311
nu = 0.7212647167050273
obj = -871.7519813717781, rho = 28.853348694450805
nSV = 125, nBSV = 86
Total nSV = 125
..*
```

(a) *Screenshot 1*

```
Identifying Potential Customers in Tweets (Martijn Oele)
Enter path to directory <Leave empty for default path>:
Reading keywords from file
Reading influencers from file
------------------------
          MENU
------------------------
 1) Crawl tweets
 2) Extract features experiment 1
 3) Extract features experiment 2
 4) Train & Test experiments
 0) Exit
------------------------
```

(b) *Screenshot 2*

```
> Results of test
    Threshold   Precision   Recall
    0.0     0.46        1.00
    0.1     0.46        0.99
    0.2     0.45        0.97
    0.3     0.46        0.97
    0.4     0.46        0.95
    0.5     0.42        0.05
    0.6     0.43        0.05
    0.7     0.29        0.01
    0.8     0.50        0.00
    0.9     0.00        0.00
    1.0     0.00        0.00
```

(c) *Screenshot 3*

Figure 3: *Screenshots of Java implementation.*

3.4 Features

Somehow, it is necessary to store some properties of the data into features. These features are used by a machine learning algorithm to make a prediction. The features are inspired by the human behaviour, since the artificial intelligence model tries to copy the human process of decision making. There are several studies that investigate how humans identify intentions in texts. Goldberg et al. (2008) did a study on how wishes can be detected. In their paper they write that we first try to identify the topic of a text. Then humans try to identify the sentiment of the text. Sentiment is the attitude of an author with regards to a certain topic. Besides the topic and sentiment of the texts I have used a third property of the tweets: the number of days between the end of the timeline and a tweet that expresses the purchase intention. Imagine two persons. The first person has posted a tweet that expresses a purchase intention for a particular product two months ago and the other person has posted a tweet that expresses a purchase intention for the same product yesterday. Intuitively, it is more likely that the second person still wants to buy the product.

I have used two different types of features. The three properties that are mentioned here form the basis of the first type of the features. These features are called abstract features and are knowledge-rich. The other type of features is knowledge-poor. I have used two knowledge-poor features: unigrams and skip-bigrams. In this field, unigrams and skip-bigrams are widely accepted as standard when you want features to process natural language (Zampieri et al., 1992). In this study, I do not process the unigrams and skip-bigrams. In future work one might pre-process the *n*-grams and for example also use trigrams.

Tri-grams need more computing power when they are not filtered before they are used to train a model. I have separated the knowledge-poor features from the knowledge-rich features there is a significant difference in the size of the features. The unigram feature has over 4000 values, whereas the abstract feature has only 9 values. When the features would not be separated, the contribution of the knowledge-rich features would be nil. Besides, when they are separated I can investigate how much the two types of features contribute to each other in terms of performance by training models on the abstract features only, on the knowledge-poor features only and on both types of features.

Tag	Description
N	Noun
^	Proper noun
V	Verb
!	Interjection
#	Hashtag
@	Username mention
~	RT @user in retweet
U	URL
E	Emoticon
$	Numeral
,	Punctuation
G	Abbreviation

Table 1: *Relevant tags from POS Tagger for Twitter*

The knowledge-poor features store all words that are used by all users in the dataset. Somehow, I needed to separate all tokens in a tweet. This is done using a Java library that offers a Part-Of-

Speech tagger especially for tweets[11]. This tagger returns all separate tokens in a tweet assigned with a tag. The most important types of tokens are shown in table 1.

3.4.1 Unigrams

This feature is the first knowledge-poor feature. The model creates a list with all unigrams that appear in all tweets of all users in the dataset. A unigram is a single token in a text. I do not store URLs in the unigram list, because of several reasons. First, URLs are shortened by Twitter, so the URL itself does not give any information. Second, the model does not read the content of the website the URL is referring to and since there are many different URLs it is not likely that the model finds patterns in the URLs. Every uppercase in a token is replaced by a lowercase and all diacritical signs are removed. Although I think diacritics are not frequently used in English texts, I have excluded the coincidence when relevant words are (accidently) spelled different. The list is shrinked by removing all unigrams that have a frequency of 1 in the entire dataset. When the frequency of a unigram is 1, there is only one user that has used this token, so this token can not be used to find a pattern. When the list of unigrams is created, the model loops over all tweets in the timeline of a user and creates a list with frequencies of all unigrams. These frequencies are divided by the number of tokens in all tweets of a user. In my Java implementation, the counting of unigrams happens on a separate thread. When the list with frequencies is created for every user, a model is trained on these frequencies. The output of the trained model is the confidence value, i.e. the certainty that an instance belongs to class A ($P(A) \in [0, 1]$). This confidence value is used as additional abstract feature `learned_unigrams`.

User	$unigram_1$	$unigram_2$	$unigram_3$	$\cdots$	$unigram_n$
$user_1$	$z_{1,1}$	$z_{2,1}$	$z_{3,1}$	$\cdots$	$z_{n,1}$
$user_2$	$z_{1,2}$	$z_{2,2}$	$z_{3,2}$	$\cdots$	$z_{n,2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$user_m$	$z_{1,m}$	$z_{2,m}$	$z_{3,m}$	$\cdots$	$z_{n,m}$

Table 2: *Matrix with unigram frequencies.*

Table 2 shows a matrix with frequencies of unigrams. For a user y, the frequencies of unigrams are normalised, i.e. $z_{x,y}$ = frequency of $unigram_x$ / #unigrams in the timeline of user y. The matrix with the frequencies of skip-bigrams has the same structure.

3.4.2 Skip-bigrams

The skip-bigrams form the second knowledge-poor feature. It is more or less the same as the unigram feature. The only difference is that it creates a list of skip-bigrams instead of a list of unigrams. A k-skip-$n$gram is a sequence of n tokens (i.e. n-gram)

[11] http://www.cs.cmu.edu/ ark/TweetNLP/

where there can be k or less tokens between the two unigrams. Guthrie et al. (2006) use the following sentence in their paper to explain skip-grams:

"Insurgents killed in ongoing fighting."

In this paper they mention that the 2-skip-bigrams are all pairs of unigrams in the sentence with a maximum of 2 tokens in between the unigrams. The set of 2-skip-bigrams in this sentence according to the paper is:

{insurgents killed, insurgents in, insurgents ongoing, killed in, killed ongoing, killed fighting, in ongoing, in fighting, ongoing fighting}

. In my study, I have used k-skip-bigrams slightly different. The value of k is different for each tweet. I have set k to be the length of the tweet -2 (i.e. the number of unigrams in a tweet). So, the skip-bigrams are all pairs of words that occur in a tweet. Consider the example sentence as a tweet in my dataset. The set of 3-skip-bigrams becomes now:

{insurgents killed, insurgents in, insurgents ongoing, insurgents fighting, killed in, killed ongoing, killed fighting, in ongoing, in fighting, ongoing fighting}

Again, the URLs are not used, uppercases are replaced by lowercases and all diacritics are removed. The list is filtered by removing all skip-bigrams that occur at most 3 times, The frequencies of all skip-bigrams in the list are counted on a separate thread for every user. Again, the frequencies are normalised. Then, the frequencies are the input for machine learning and the output is a additional abstract feature: `learned_skipbigrams`.

3.4.3 Maximum relevance score

For all tweets in a timeline, the model computes a relevance score. This score tells how relevant a tweet is with regards to the product that is used. In order to compute this score, I have used a list of product-related and purchase related keywords. Note that these keywords are not the same as the keywords that were used to collect tweets. Most keywords are given by the human expert. These keywords have a weight that is denoted by the human expert. Other keywords are based on the product's website and the product's Facebook page. The weights of these keywords are determined by looking at how often this word is used on the webpages. The complete list of keywords can be found in appendix I. When a tweet contains a keyword, it's relevance score is incremented with the weight of a keyword. Every relevance score of a tweet that is computed in the algorithm and is normalised by dividing the score over the number of tokens in that particular tweet. Tweets with a relevance score higher than 0 are stored as relevant tweets. The `max_relevance_score` of a user is the highest relevance score in the timeline of that user. When the annotator was able to make a decision before the end of the timeline, it is possible that the most relevant tweet is not really the most relevant tweet, because the annotator has assigned a label before the most relevant tweet of the timeline had been posted.

3.4.4 Days since maximum relevance score

This feature returns the number of days between the tweet with the maximum relevance score and the end of the timeline. The rationale behind this feature is that the probability of a user being a potential customer is higher when he recently used relevant terms. The number of days is divided by the total length of the timeline of the user. The length of the timeline depends on the position where the annotator has assigned a label. The name of this feature is `days_since_max_relevance_score`.

3.4.5 Sentiment of most relevant tweet

The sentiment of the tweet with the maximum relevance score is computed and stored in the feature: `sentiment_tweet_max_relevance_score`. I want to measure the sentiment of the most relevant tweet because whether this tweet is positive or negative can make a difference in the final prediction. The sentiment of a tweet is obtained by the Sentiment140 API. This API takes a tweet and a topic as input and returns -2 (negative), 0 (neutral) or +2 (positive). It does not use additional settings.

3.4.6 Average relevance score

This features computes the average relevance score of all tweets in the timeline. When this score is high, the user actively talks about the product or uses related keywords or posts tweets that are very relevant. The name of this feature is `avg_relevance_score`.

3.4.7 Average sentiment of relevant tweets

This feature computes the average sentiment of all tweets with a relevance score > 0. It is important to know if the user is positive or negative when he uses relevant terms in his tweets. This feature is called `avg_sentiment_relevance_tweets`.

3.4.8 Relevance score of last relevant tweet

I want to know the relevance of the last relevant tweet of the user. If someone recently posted a very relevant tweet, he is more likely to be a potential customer at the end of the timeline. The name of this feature is `relevance_score_last_relevant_tweet`. Since the relevance scores have been normalised already, it is not necessary to normalise the output of this feature again.

3.4.9 Days since last relevant tweet

Together with the previous feature the algorithm stores how relevant someone's tweet was and how many days ago it has been posted. This feature is called

days_since_last_relevant_tweet. When this number is low, the possibility that the user is a potential customer increases. Again, the number of days is divided by the length of the timeline of this user.

3.4.10 Sentiment of last relevant tweet

The sentiment of the last relevant tweet is important to know, since a very relevant tweet can also be negative. When this tweet is negative, the author is less likely to be a potential customer. The value of the sentiment is stored in the feature sentiment_tweet_last_relevant_tweet.

3.4.11 Following company and influencers

At last, I have created a feature that computes a score that represents the user's involvement with the company. To compute the involvement score I have again used a list of weighted Twitter accounts. The human expert gave me a list of Twitter accounts of possible influencers. I have extended this list by adding the Twitter account of the product (tweethue) and Twitter accounts of other departments of Philips. The weight of an influencer represents the importance of this account. The list of influencers can be found in appendix II. For each of these accounts, I check if the user follows this account. If so, the following_influencing_accounts feature is incremented with the weight of the account that is followed by the user. It is not possible to check whether a user was followed at a specific date. So, this feature is time-independent and the values are the same in both experiments (experiments are explained later).

3.5 Machine Learning

Data mining is the extraction of information (knowledge) from massive datasets (Way, 1996). Extracting information is usually done by applying machine learning to those datasets. Machine learning is concerned with minimizing the error of a model that can be used to find 'hidden' patterns in datasets. Usually, such a model is trained until the error can not be decreased anymore. The patterns that are found are used to make predictions (classification), to divide the data into groups (clustering) or to detect outliers (anomaly detection). The datasets that are used are often very large, so potential relationships between variables cannot be found by a human analyst. There exist many different machine learning algorithms, each with its own strengths and weaknesses. In this study I have used machine learning to make predictions. I have decided to apply machine learning to the unigram feature, to the skip-bigram feature, to the abstract features and to all features combined. In this study I have used supervised machine learning algorithms, so the algorithms use true classes to train a model. For example, when a model is trained on the unigram features, a machine learning model tries to find a pattern in the frequencies of words that are only used by users that are assigned as potential customer.

There are different properties of machine learning algorithms that can influence the output of the algorithm. First, the size of the training set and the number of features (input dimensions) are important to know. Some machine learning algorithms are easy to scale for training on larger datasets, while for others the training time exponentially grows when the input size increases. Second, some algorithms often have a lower accuracy. However, they can be quite faster in training than others. There should be made a trade-off between training time and accuracy. So, in this section I compare multiple algorithms with their strengths and weaknesses and explain which algorithms I have used.

Linear Regression
Linear regression is a statistics technique. This technique assumes that there is a linear relationship between variables in the dataset (input) and the label of an instance (output). Linear regression tries to find a linear function that can compute the output with the input ($y = B_0 + B_1 * x$). This algorithm does not use parameters, so tuning is not necessary. I have used linear regression as baseline, since it is a very simple and standard technique. The performance of other algorithms are compared to the performance of linear regression.

Decision Tree
Classification Tree
A classification tree is a top-down machine learning algorithm that can be interpreted without much knowledge of machine learning Safavian and Landgrebe (1991). The labels for test instances are assigned by walking through the tree. Every attribute of the data has a node in the tree and every possible value has a leaf that leads to another node. When the attribute has continuous values, the algorithm computes the optimal threshold that is used to split the data into two (or more) subsets. The decision tree algorithm tries to create a tree that partitions the data into subsets with all instances having the same label. Figure 4 shows a simplified version of a decision tree for this project.

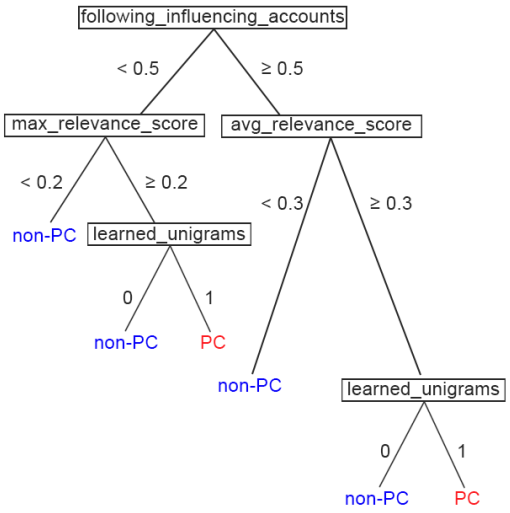


Figure 4: *Simplified Decision Tree*

There are a couple of metrics that can be used to compute the variance within a subset to partition the data. Some of them use the entropy (the degree of uncertainty). An advantage of this algorithm, in contrast to other algorithms, is that it can handle both categorical and numerical attributes. Furthermore, decision trees work for two-class data as well as for multiple class data and they can handle large datasets (note that when there are many attributes the trees become very complex and the result tends to be inaccurate). Another disadvantage of implementations of decision trees is that they handle missing values in a variety of ways. Some implementations assign all instances with missing values to the node that has the most instances. Other types assign these instances to a random node. Since the output of decision trees is easily affected by small changes in the data, the accuracy is not always as high as possible. I think that decision trees are not the best option for training models on the frequencies of the unigrams and skip-bigrams, since there are too many attributes and the variance within the attributes is very small.

Random Forest

Random forest combines multiple decision trees. The output is the mode of the outputs of all decision trees. Random forest is used to compensate for the fact that a single decision tree is very sensitive to small changes in the data. Furthermore, decision trees tend to overfit the data and random forest can correct for this. Random forest are less easy to interpret, but the variance within the partitions becomes smaller, while the performance is usually better. The random forest algorithm applies bagging: it selects random subsets of the data and creates a tree that fits this subset best. This process is repeated x times, after which the outputs are averaged. Depending on the size of x, this algorithm can be fast and it almost works off-the-shelf. Random forest is not the best option to train models on the unigram and skip-bigram frequencies because there are too many attributes and the variance within the attributes is still very small. However, it can be a good model to train the abstract features. The advantage of using this model is that one can easily detect which features are most important.

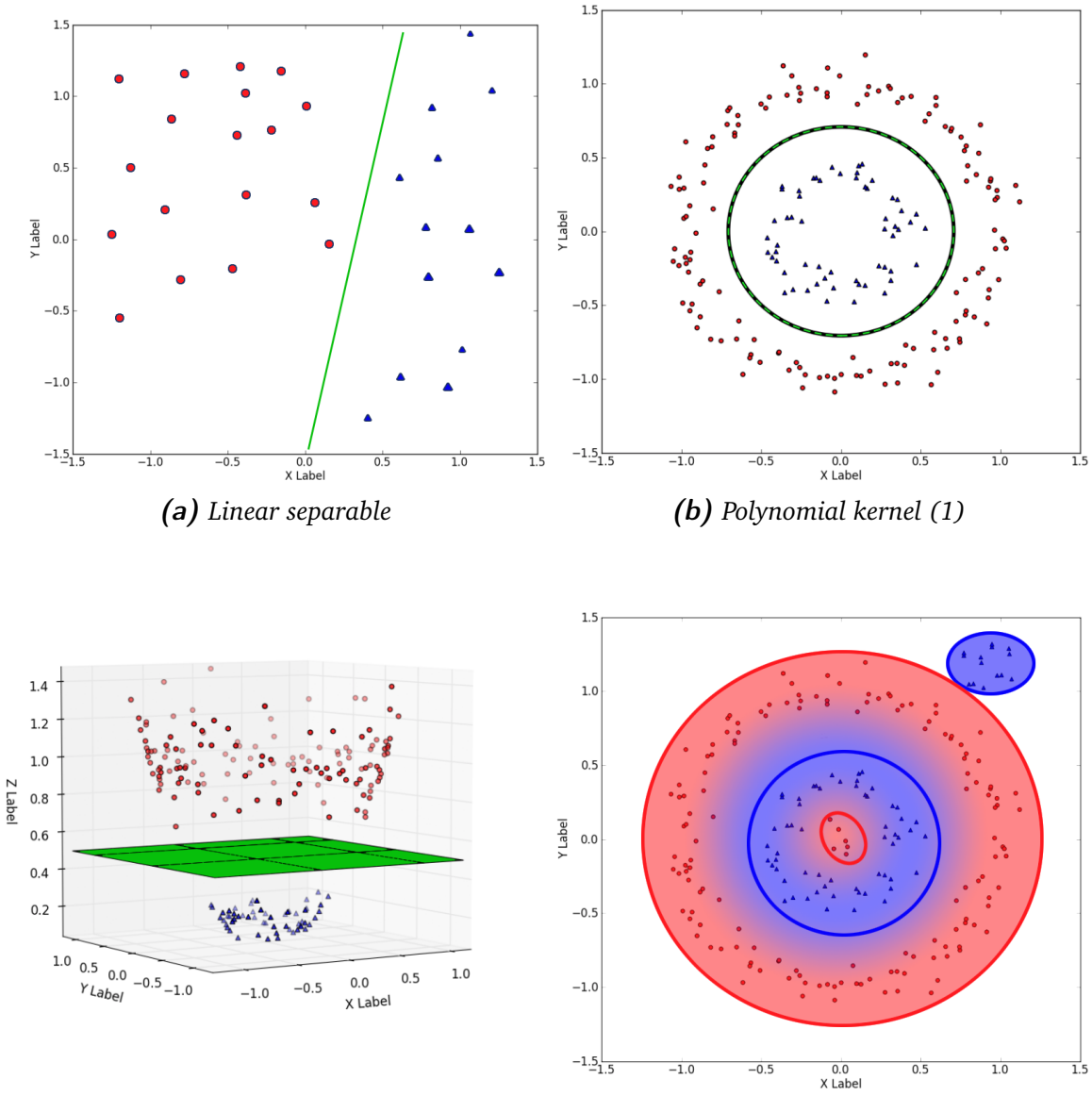
Support Vector Machine
A support vector machine (SVM) is a supervised model that represents data instances as points in a multidimensional space (Andrew, 2000). This model is less easy to interpret compared to decision trees. However, it can handle missing values by just ignoring them. SVMs tries to create a linear gap that divides the space into two parts. This gap should be as wide as possible. It tries to separate the instances belonging to one class from instances belonging to the other class. A support vector machine works for two-class datasets only, since it tries to assign an unknown instance to either one or another part of the space. The input for an SVM is represented as vectors of features. A so-called kernel function maps these vectors to a space. There are several kernel functions. I will briefly explain the kernel functions that I have used. At the end, some of the vectors that are mapped are assigned as support vectors, i.e. vectors that are used to find the line that separates the two classes.
When you use an SVM, you should spend more time on pre-processing the data. This is usually done by scaling the data. For each feature, the mean should be zero and the standard deviation should be 1, so all points are between -1 and +1. SVMs are ideal when you need to categorise text data or images. To classify this type of data that has a huge number of features, other machine learning algorithms need a lot more time for training. Therefore, an SVM may be a good choice to classify the unigrams and skip-bigrams.

Linear kernel
A linear kernel is usually faster than any other kernel. It tries to find a linear plane or line that separates the data (see Figure 5a). A linear kernel uses just one parameter that should be tuned. This parameter C is a measure for how much you want to avoid that an instance is misclassified. When C is high, the cost of misclassification is also high. When C is low, the cost of misclassification is low. You need to find a balance between being too strict and being too loose.
An SVM with a linear kernel tries to find some function $K(x, y) = x * y$ that translates the data points in such a way that they are linearly separable. A linear kernel is recommended when classifying text data, since this type of data is often linearly separable due to the high number of features.

Polynomial kernel
When data is not linearly separable, you can decide to use a polynomial kernel. This kernel uses more parameters than a linear kernel. Now you need to tune C as well as a parameter γ. This parameter is a measure for the sensitivity to differences in the features. The default value is $\frac{1}{\#features}$. A polynomial kernel tries to map the points in the feature space to a higher dimension with a function of the form $K(x, y) = x^T y + c)^d$ where d is the degree of the kernel. Figure 5b and figure 5c show how a non-linear separable problem can be mapped onto a space with more dimensions. In this new space, the points can be separated by a hyperplane. Finding this hyperplane is less difficult for a machine compared to finding the function that separates the points in Figure 5b. Therefore, choosing the right kernel function is a very important step when building a support vector machine.

RBF kernel

Another type of a kernel is the radial basis function (RBF) kernel. The most popular RBF kernel is the Gaussian RBF kernel. This kernel tries to form circles about all data points, where all points that fall within a particular circle have to belong to the same class. It uses the function $K(x, y) = exp(-\gamma * |x - y|^2)$ where γ is the strength of the influence of a single training instance. Figure 5d shows an example of how an SVM with an RBF kernel classifies a dataset.



(a) *Linear separable*

(b) *Polynomial kernel (1)*

(c) *Polynomial kernel (2)*

(d) *RBF kernel*

Figure 5: *Examples of svm classification with different kernels. (a) Input vectors that are separated by a linear kernel. (b) Input vectors that can not be separated by a linear kernel. (Kim, 2013) (c) The space is transformed using a polynomial kernel, so the instances can be separated (Kim, 2013) (d) Input vectors that are separated by an RBF kernel.*

Sigmoid kernel

A sigmoid kernel uses the sigmoid function from linear regression ($K(x, y) = tanh(\gamma * x^T y + r)$). An SVM that uses a sigmoid kernel is equivalent to a two-layer neural network. The sigmoid kernel is only conditionally positive definite (Lin and Lin, 2003), so it is not widely used. The result of classifying using a sigmoid kernel is similar to the result of classifying using an RBF kernel (figure 5d).

Neural Network

A neural network is a machine learning algorithm that is based on a biological neural network. It can classify non-linear separable datasets. Another advantage of neural networks is that they can become as large as needed, but the computation will not increase exponentially. However, a neural network is more like a black box, so it difficult to adapt the network when you want better performance.

The aim of a neural network is to find a function $f : X \rightarrow Y$ that maps a data point $x \in X$ to it's label $y \in Y$. The algorithm tries to minimise the difference between the real label y and the network's output $f(x)$, which is usually measures by the mean-squared error. A reliable neural network has many 'neurons' and uses multiple layers, but one will need a large dataset to train such network. Since my dataset might be too small, I did not choose to build a neural network, although it might be a good option for this kind of data.

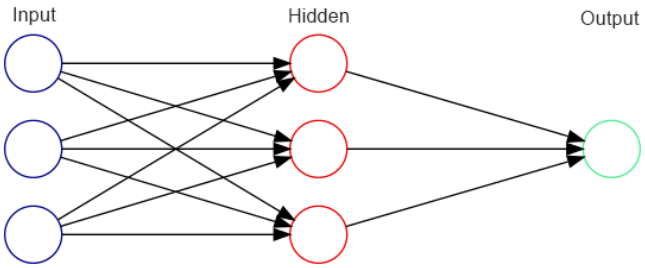


Figure 6: *Example of a Simple Neural Network*

Figure 6 shows the structure of a simple network. It can contain multiple hidden layers. After every layer, a value for each class is increased or decreased and at the end, a test instance is assigned to the class that has the highest value.

Naive Bayes Classifier

A naive Bayes classifier is an algorithm that is based on Bayes' theorem. The features are assumed to be independent once the class labels are known. The output is computed via the function $\hat{y} = \arg \max_{k \in \{1,\ldots,K\}} p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$.

A naive Bayes classifier, a popular algorithm for classifying text documents, uses a couple of parameters that are linear in the number of instances and features. However, the training time of this machine learning model increases linearly with the size of the dataset. So, I chose not to use a naive Bayes classifier, since it would take a lot of time to train, update parameters for a better performance, train again and so on.

4 Experiments & Results

I have conducted two experiments. In the first experiment, the model uses all tweets in the users' timelines. In the second experiment, the model only uses the tweets until the position where the annotator has assigned the true label. Table 3 show some properties of both experiments.

	Experiment 1	Experiment 2
Number of samples	214	
#positive instances	101	
#negative instances	113	
Number of unigrams	5,820	2,028
Number of skip-bigrams	60,055	20,447
Number of abstract features	11	11

Table 3: *Statistical properties of experiments.*

In each experiment I have trained four different models. One model is trained on nine abstract features, another model is trained on the unigram frequencies, yet another is trained on the skip-bigram frequencies and the last model is trained on 11 abstract features (nine original abstract features + `learned_unigrams` + `learned_skipbigrams`). The `learned_unigrams` and `learned_skipbigrams` features are confidence values. These four models are trained using the machine learning techniques that I have explained in the previous section, except for neural networks because a good neural network uses multiple layers, but is hard to train and needs a lot of tuned parameters.

I have used Weka (Frank and Witten, 2016) to train models with 10-fold cross-validation. k-fold cross-validation is used when the dataset is too small to use a fixed part to train a model and use the rest of the data for testing. The dataset is split up into k samples. $k-1$ samples are used to train a model. The k^{th} sample is used for validation. I have set the random seed to 0, so the subsamples are not randomly created. I have used this setting because I need to know to which instance a confidence value belongs in order to add the additional features to the abstract features. Table 4 shows the parameters that are used for the support vector machines. I have used Weka's default values, since Weka does not have a function that can tune parameters automatically. There are too many combinations of parameter settings, so tuning them manually would need too much time.

Parameter	Value
$coef_0$	0
cost	1
degree	3
eps	0.001
γ	0
loss	0.1
nu	0.5
shrinking	true

Table 4: *Parameters that are used for support vector machines.*

Weka has returned the confidence values instead of either one class or the other class. After each run I have stored the confidence values. For classification models that use nominal classes (SVMs, naive Bayes and random forest), it also returns the area under the precision-recall curve. The precision-recall curve is formed by computing precision and recall for several thresholds. Usually, the threshold is set to the confidence values of all instances. The linear regression model uses numeric classes, so the precision-recall curve can not be created. I have used the output of the linear regression model to create the curve manually. The results are not probabilities between 0 and 1, so I have normalised the output before I have formed the precision-recall curves. Then I have approximated the area under those curves. Table 5 and table 6 show the area under precision-recall curve for all combinations of input and algorithm.

$\textbf{Input}_{\#features}$	LR	NB	RF	lin-SVM	poly-SVM	rbf-SVM	sig-SVM
$\textbf{Abstract}_9$	0.467	**0.580**	**<u>0.605</u>**	0.558	0.499	0.558	0.489
$\textbf{Unigram}_{4000}$	0.521	0.579	0.602	0.498	N/A	N/A	N/A
$\textbf{Skipgram}_{4000}$	0.414	0.482	0.506	0.522	N/A	N/A	N/A
$\textbf{All}_{11}$	**0.538**	0.568	0.577	**0.595**	0.528	0.532	0.553

Table 5: *Experiment 1: Area under precision-recall curve for combinations of input and machine learning algorithm. Bold values denote the optimal input for an algorithm. For the SVM I have denoted only one highest value for all kernels. The underlined value denotes the best combination of input and algorithm. LR = Linear Regression; NB = Naive Bayes; RF = Random Forest; lin-SVM = SVM (linear kernel); poly-SVM = SVM (polynomial kernel); rbf-SVM = SVM (RBF kernel); sig-SVM = SVM (sigmoid kernel).*

Input$_{\#features}$	**LR**	**NB**	**RF**	**lin-SVM**	**poly-SVM**	**rbf-SVM**	**sig-SVM**
Abstract$_{9}$	0.595	0.518	0.543	0.484	0.501	0.505	**0.544**
Unigram$_{2028}$	0.506	**0.534**	0.531	0.525	N/A	N/A	N/A
Skipgram$_{4000}$	0.566	0.504	0.533	0.488	N/A	N/A	N/A
All$_{11}$	**0.595**	0.505	**0.574**	0.521	0.534	0.502	0.519

Table 6: *Experiment 2: Area under precision-recall curve for combinations of input and machine learning algorithm. Bold values denote the optimal input for an algorithm. For the SVM I have denoted only one highest value for all kernels. The underlined value denotes the best combination of input and algorithm. LR = Linear Regression; NB = Naive Bayes; RF = Random Forest; lin-SVM = SVM (linear kernel); poly-SVM = SVM (polynomial kernel); rbf-SVM = SVM (RBF kernel); sig-SVM = SVM (sigmoid kernel).*

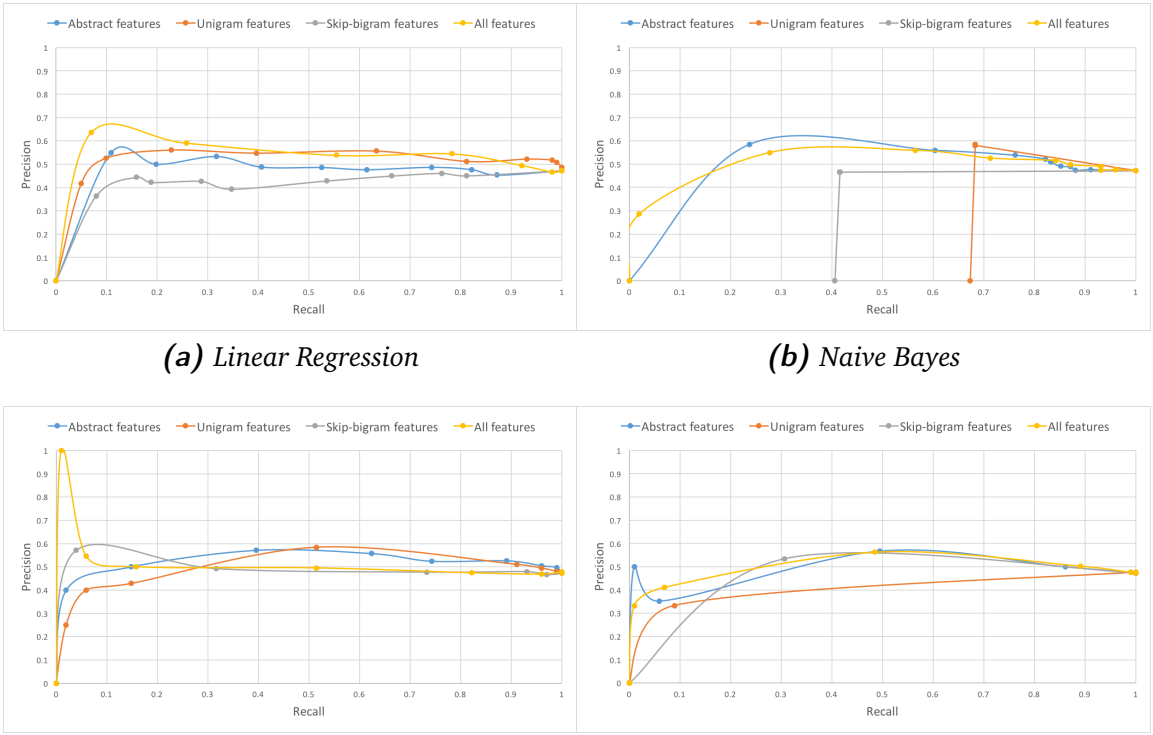
Table 7 shows how fast models could be trained and validated on a relative scale. There was no significant difference between the durations in the two experiments.

Input	**LR**	**NB**	**RF**	**lin-SVM**	**poly-SVM**	**rbf-SVM**	**sig-SVM**
Abstract	-	+++	+++	-	+	++	++
Unigram	—	++	++	–	N/A	N/A	N/A
Skipgram	—	++	++	–	N/A	N/A	N/A
All	-	+++	+++	-	+	++	++

Table 7: *Time that was needed to train and validate all models on a relative scale (—, –, -, +, ++, +++). LR = Linear Regression; NB = Naive Bayes; RF = Random Forest; lin-SVM = SVM (linear kernel); poly-SVM = SVM (polynomial kernel); rbf-SVM = SVM (RBF kernel); sig-SVM = SVM (sigmoid kernel).*

Figure 7 shows the precision-recall curves of each input in experiment 2. Figure 7a shows the curves for a linear regression model. Figure 7b shows the curves for a naive Bayes model. Figure 7c shows the curves for a random forest model. Figure 7d shows the curves for a support vector machines model. For the SVM I have only used the kernel with highest area under the curve. The highest area under the curve for the abstract features was obtained when classifying with a linear kernel. The highest area under the curve for all features (9 abstract features + 2 additional features) was also obtained with a linear kernel. I have only trained SVM models on the unigram and skip-bigram features using a linear kernel, because a linear kernel should be the best option when the number of features is much higher than the number of samples.

The curves of the unigram and skip-bigram input for naive Bayes are different than all other curves, because the output of the naive Bayes classifier was almost always 0 or 1 and in a few cases very close to 0 (< 0.010) or very close to 1 (> 0.990). Most of the thresholds had no effect on the precision and recall values, so the curves is formed by only a few of combinations of precision and recall.

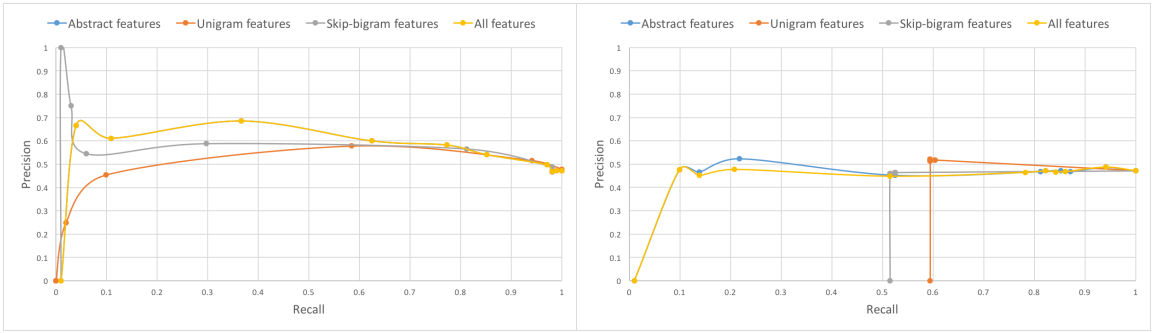


(a) *Linear Regression* ***(b)*** *Naive Bayes*

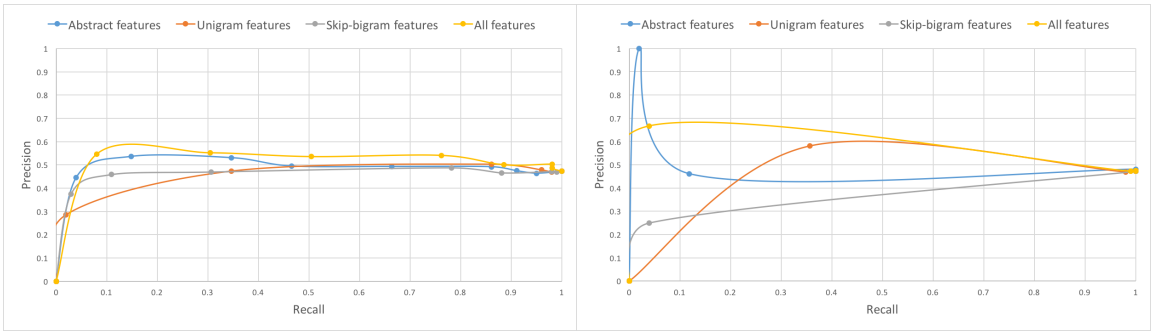
(c) *Random Forest* ***(d)*** *Support Vector Machine*

Figure 7: *Within-algorithm precision-recall curves of experiment 1. The subplots show the precision-recall curves of different inputs for (a) linear regression, (b) naive Bayes, (c) random forest and (d) support vector machines.*

Figure 8 shows the precision-recall curves of each input in experiment 1. Figure 8a shows the curves for a linear regression model. Figure 8b shows the curves for a naive Bayes model. Figure 8c shows the curves for a random forest model. Figure 8d shows the curves for a support vector machines model. For the SVM I have again only used the kernel with highest area under the curve. The abstract features have the highest area under the curve when a sigmoid kernel is used. Again, the unigram and skip-bigram features are only used to train a model with a linear kernel. When two additional features are added to the abstract features, an SVM with a polynomial kernel has the highest area under the curve. The curves of the unigram and skip-bigram input for naive Bayes are again different than all other curves, because of the same reason in experiment 1.



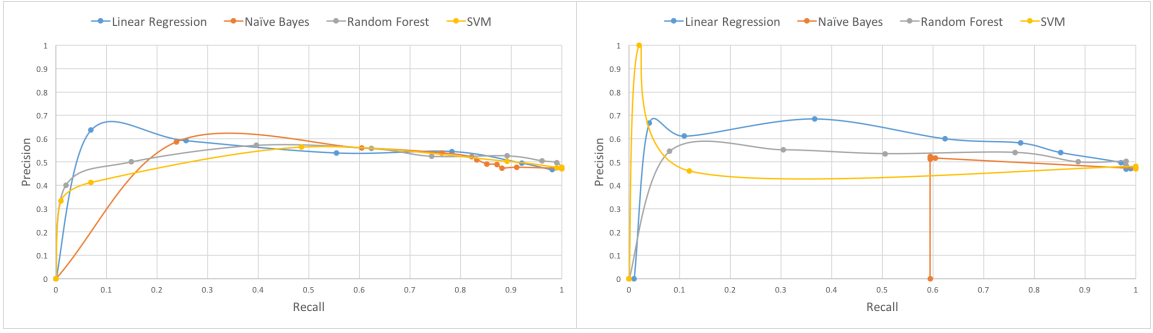
(a) *Linear Regression* ***(b)*** *Naive Bayes*



(c) *Random Forest* ***(d)*** *Support Vector Machine*

Figure 8: *Within-algorithm precision-recall curves of experiment 2. The subplots show the precision-recall curves of different inputs for (a) linear regression, (b) naive Bayes, (c) random forest and (d) support vector machines.*

I have used the values from the tables above to determine which algorithm performs best in an experiment (bold values in table 5 and table 6). Figure 9a and figure 9b show the between-algorithm precision-recall curves for experiment 1 and experiment 2.



(a) *Experiment 1* ***(b)*** *Experiment 2*

Figure 9: *Between-algorithm precision-recall curves of (a) experiment 1 and (b) experiment 2.*

Figure 10 shows the precision-recall curves of the best combinations of input and algorithm of both experiments (underlined values in table 5 and table 6).

Figure 10: *Precision-recall curve of the best combinations of input and algorithm for both experiments.*

5 Discussion

The interpretation of the results can be different for the type of product and the goal of the selling company. A company might want to send targeted advertisements to people that are classified as a potential customer. In this scenario, it is preferred to have the number of false negatives as low as possible because a company does not advertise to all potential customers. A high number of false negatives will result in less new customers for the company. One might argue that people will go to the company although they do not receive advertisements. However, small or new companies do not have the advantage of being a company with a good reputation. In another scenario a company wants to invest in its new potential customers. In this scenario, it is not preferred when there are many false alarms because that would be a waste of investments. Therefore, I have used the precision-recall curves as main result, because they do not depend on the need for high precision or high recall.

The area's under the curves in all figures are not equal to the area's under the curves from table 5 and table 6. Weka uses 214 thresholds to compute 214 different precision-recall combinations. Weka can return the precision-recall curves, but it can not plot multiple precision-recall curves in one figure. Therefore, I have used a consistent series of thresholds for all outputs $(0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0)$. At each threshold, a class is assigned to an instance (*PC* if the confidence value $>=$ threshold and *non-PC* otherwise). Then, I have computed the number of true positives (hits), false positives (false alarm) and false negatives (prediction $=$ *non-PC* but actual class $=$ *PC*). The precision and recall can be computed with these numbers. For each combination of input and algorithm, I have used 11 combinations of precision and recall values to create a curve. The area under the curve that can be approximated from this curve is less accurate then the area under the curve from the results in Weka, because the curves I have created use less points than the plots that are formed by Weka. Therefore, the area's under the curves in the figures are smaller than the area's under the curves in the tables. In this section, I will use both the tables with Weka's results and

the curves that I have plotted using 11 different thresholds to compare the performance of algorithms, where the performance is expressed in terms of area's under the curve. So, the higher the area under the curve, the higher the performance of a model.

Within-algorithm comparison
First, I compare the results for different inputs in experiment 1 (table 5). For naive Bayes and random forest is shown that the area under the curve (AUC) of abstract features as input and the AUC of unigrams as input are almost equal. However, the AUC becomes smaller when the two additional abstract features are added. The table shows that the AUC of naive Bayes with skip-bigrams as input is lower than 0.5 and since the AUC decreases when all features are used to classify the data, I conclude that the skip-bigrams add noise to the abstract features. Therefore, I have applied naive Bayes and random forest to the abstract features with the `learned_unigrams` only. For naive Bayes, the AUC is even lower (0.566) than the AUC for all features. For random forest, the AUC is higher than the AUC for all features and just slightly lower than the AUC of the abstract features (0.596). These results support my conclusion that skip-bigrams add noise when classifying with random forest or naive Bayes.
For all types of kernels in an SVM, the AUC increases when the additional features are added to the abstract features. The results of linear regression show the highest increase of the AUC when the additional features are added. The abstract features have an AUC of 0.467. This is surprising because I expected that there is a linear relationship between the abstract features and the class. However, I can think of an explanation. The selection of keywords that are used to compute the relevance score might be too wide, so users with the *non-PC* label also have a high relevance score. Then, the linear relationship weakens and a linear regression model can not classify the data.

In experiment 2 (table 6), naive Bayes performs worse when all features are used instead of the abstract features. Again, the AUC of the skip-bigrams is around 0.5, so it is likely that the skip-bigrams add noise. For random forest, there is less difference between the AUCs of the abstract features and the skip-bigram frequencies compared to experiment 1. Now, the AUC becomes bigger when the additional features are used. For the support vector machine, the conclusions from experiment 1 do not hold in experiment 2. With an linear and polynomial kernel, the AUC becomes higher when all features are used. With the other kernels, the AUC becomes lower. A polynomial kernel has always a higher AUC in experiment 2 than in experiment 1, but the difference is very small. The other kernels always perform worse in experiment 2. This shows that an SVM works better if all data can be used instead of just a subset.

Between-algorithm comparison
When the input consists of abstract features, almost all algorithms perform better than chance level (precision $> 50\%$), except for the support vector machine and linear regression in the first experiment. In experiment 1, the highest AUC is obtained when classifying with a random forest model. In experiment 2, linear regression performs best. From the tables can be concluded that if the difference between the AUC of skip-bigrams and the AUC of the abstract features is smaller than 0.1, then the performance of the algorithm increases when the additional features are added to the abstract fea-

tures. The AUC of a support vector machine with a polynomial, RBF or sigmoid kernel is lower than 0.5. When using all features (abstract features + 2 additional features), random forest performs also well, but an SVM with a linear kernel is slightly better (AUC of 0.577 and 0.595 respectively).

The results of support vector machines are less consistent with my expectations. I expected that either a polynomial kernel, an RBF kernel or a sigmoid kernel would perform better on the abstract features compared to a linear kernel, because the number of features is very small. Nevertheless, for both the abstract features and all features, a linear kernel has the highest AUC. When the knowledge-poor features are used as input, an SVM with a linear kernel performs worse than other algorithms.
According to figure 9a there is no algorithm that always has the highest precision for any value for recall. When recall is higher than 0.5, the precision is more or less the same for algorithms. In the left half of the graph, the linear regression curve is much higher than all other curves. So, when you do not need high recall, linear regression is the best option. In the second experiment (figure 9b), the linear regression curve is almost always higher than any other curve. For very low values of recall, the support vector machine has a very high precision, but this curve decreases exponentially after the first threshold. Random forest is slightly worse than linear regression, but it is still better than support vector machine and naive Bayes.

Between-experiment comparison
Figure 10 shows the curves belonging to combinations of input and algorithm with the highest AUC of experiment 1 and experiment 2. This figure shows that the precision is always higher (for any value for recall) in experiment 1. So I can conclude that the machine learning methods perform better when the features are extracted from more tweets. This is consistent with the idea that the results of machine learning become more accurate when you use more data to extract features.

6 Conclusion

This section is mainly focussed on answering the research question. I will also look back to this project and mention some limitations and restrictions. First, it was necessary to adapt the research question. Initially, I wanted to make an artificial intelligence model that can identifies potential customers based on tweets. It was hard to find a ground truth for this study. The output of this model could not be compared to the predictions assigned by an annotator, because this annotator is also not sure if someone is a potential customer. The dataset should consist of actual customers of a company and people that will not be customers of this company. However, companies do not just give a list of customers due to privacy reasons. Besides, this type of dataset is not yet publicly available. I changed the research question, so I was going to make a model that predicts whether the author of tweets can be a potential customer or not.

Second, I wanted to implement my own machine learning model. I was only able to test this model to a small dataset in the first months, because not all data was an-

notated yet. I received the complete dataset just within days before the deadline, so this was a bit problematic. I observed that the results of classifying the entire dataset were not good. Therefore, I have used Weka to use other algorithms (instead of my own SVM implementation and without tuning), so I was able to compare results from multiple algorithms.

Thirdly, the curves do not correspond to the AUCs in the tables. This is caused by the fact that I have used other thresholds to compute the precision and recall. I have used the curves that I have created to compare performance of algorithms in different experiments, but I will use the AUCs from Weka to answer the research question.

The results show that there is no algorithm that is significantly better than other algorithms when Twitter data is used for classification. The AUCs range from 0.538 to 0.605 in the first experiment and from 0.534 to 0.595 in the second experiment. In both experiments, 2 out of 4 algorithms perform better when all features are used. There is only one algorithm (naive Bayes in experiment 2) that performs best on the knowledge-poor features. In both experiments, the performance does not significantly increase when the unigrams and skip-bigrams are added to the abstract features. In some cases, the performance decreases when the unigrams and skip-bigrams are added. I expected that the performance of polynomial, RBF and sigmoid kernels would increase when the number of features is increased, but that is only the case for the polynomial kernel in both experiments and for the sigmoid kernel in experiment 1. A possible explanation for these observations is that the knowledge-poor features offer too few information. There are too many tokens that are used too little. The frequencies of most tokens are the same, so it is very hard to find a strong pattern. In the second experiment when there are less unigrams and skip-bigrams, the knowledge-poor features contribute in a positive way.

For most algorithms I can conclude that they perform better when they can use more data, except for linear regression. I think this makes sense because the linear relationship between attributes and output dissolves when the model uses more data to compute the features. Usually, when the AUC is 0.5, the precision is 0.5 for each value of recall, i.e. the performance is as good as random guessing. In this study, the precision of the best models for each algorithm is always higher than 50%. I can compare this to the study of Hamroun et al. (2015) where they analyses customer intentions using semantic patterns. The precision in their study was 55.59% for a recall of 55.28%. I do not know the precision for other values of recall, but based on these numbers, the performance of the models that are trained in this study is more or less the same as the performance of the model of Hamroun et al. (2015). Gupta et al. (2014) have created a model that tries to identify purchase intentions from social posts on online fora. The performance of their study is expressed in terms of area under the ROC curve (0.89). Without knowing all results I cannot compare the performance of their model to the performance of the models in my study.

To conclude, an artificial intelligence approach is not as good in predicting if someone is a potential customer compared to human annotator. The best AI model is obtained

by training on all data and using a random forest classifier to make the predictions. A t-test for proportions shows that the performance of this model is not significantly higher than the performance of the best model that is trained on a subset of the data: $p = 0.64552$, so $p > 0.05$ ($Z = 0.4564$). However, the artificial intelligence models are much faster than a human expert. Most models were trained within one minute, whereas the human annotator needed days to annotate the data. Besides, a human annotator can become drowsy after reading many tweets. An artificial does not have this disadvantage and can keep training models as long as we want. The models perform better than random guessing. So, when we improve the features and tune the parameters of the machine learning models, AI can approach the performance of a human expert to a higher extend.

7 Recommendations

This study can be extended in order to obtain more reliable results. One major improvement would be to deploy labels of users in a dataset that are not assigned by a human expert to be potential customers, but users in the dataset being actual customers. Instead of working with data from users that have used one or more keywords in their tweets, one could take users who are known to have bought a specific product. Such could be possible with the cooperation of a company having a list of customers of one or more particular products. The customers should be offered a discount or voucher if they give permission to use the tweets from a period of time before they have purchased the product. This approach offers a ground truth for my original research question, since the true classes are not based on predictions anymore but on the fact that one is an actual customer.

Another issue that applies to my study as well as to the proposed extended study, is the need for more time and computing capacity. The model starts by extracting features from all tweets and by training a model. Applying this model to the test set should be done in small steps in further research. The testing phase should start with just the oldest tweet that is posted in the period that is used. The model returns a possibility of a user belonging to either the*PC* class or the *non-PC* class. If this possibility is too low, the model should be applied again, but now it should add the next tweet on the timeline to the test set. This machine learning model can be really useful to companies when it is applied in real-time, since it detects customers as soon as possible. This however was not feasible for my project as testing the model would require more time than allowed by the project.

Finally, it would be an interesting study to investigate how important abstract features are for the classification. Initially, I wanted to experiment with the number of features that is used in this project, but all features were correlated, and randomly leaving out some features is not a good approach. However, in another study one might experiment with this. First, for each feature two separate models should be trained. One model should be trained with a single feature and the other model should be trained with all features except that single feature. This does not only show the importance of

the feature, but also its contribution to the other features.
Another approach to investigate which features are most important is to test all possible combinations of features. This will need a lot more time and cannot be done using a single device, since all the steps that I have described in my experiments have to be repeated $2^n - 1$ times where n is the number of features. In my experiments that means that all steps have to repeated 2047 times. However, when you know the most important features and a model can be trained with just these features, it will reduce the training time and increase the performance. Another approach to investigate the importance of a feature is by training a decision tree model. This model will learn which features can be used best to efficiently split the dataset.

References

Adedoyin-olowe, M., Gaber, M. M., and Stahl, F. (2014). A Survey of Data Mining Techniques for Social Network Analysis. *International Journal of Research in Computer Engineering and Electronics*, 3(6):1–8.

Alnawas, I. and Altarifi, S. (2016). Exploring the role of brand identification and brand love in generating higher levels of brand loyalty. *Journal of Vacation Marketing*, 22(2):111–128.

Andrew, A. M. (2000). An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods by Nello Christianini and John Shawe-Taylor, Cambridge University Press, Cambridge, 2000, xiii+ 189 pp. *Robotica*, 18(6):687–689.

Anta, A. F., Morere, P., Chiroque, L. N., and Santos, A. (2013). Sentiment analysis and topic detection of Spanish Tweets: A comparative study of NLP techniques. *Procesamiento de Lenguaje Natural*, 50:45–52.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27.

Finin, T., Murnane, W., Karandikar, A., Keller, N., Martineau, J., and Dredze, M. (2010). Annotating Named Entities in Twitter Data with Crowdsourcing. *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010(January):80–88.

Frank, E., H. M. A. and Witten, I. H. (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques". *Morgan Kaufmann*, 4.

Gattani, A., Doan, A., Lamba, D. S., Garera, N., Tiwari, M., Chai, X., Das, S., Subramaniam, S., Rajaraman, A., and Harinarayan, V. (2013). Entity extraction, linking, classification, and tagging for social media. *Proceedings of the VLDB Endowment*, 6(11):1126–1137.

Go, A., Bhayani, R., and Huang, L. (2009). Twitter Sentiment Classification using Distant Supervision. *Processing*, 150(12):1–6.

Goldberg, A. B., Fillmore, N., Andrzejewski, D., Xu, Z., Gibson, B., and Zhu, X. (2008). May All Your Wishes Come True : A Study of Wishes and How to Recognize Them.

Gupta, V., Varshney, D., Jhamtani, H., Kedia, D., and Karwa, S. (2014). Identifying purchase intent from social posts. *Proceedings of the 8th International Conference on Weblogs and Social Media (ICWSM 2014)*, pages 180–186.

Guthrie, D., Allison, B., Liu, W., Guthrie, L., Wilks, Y., and Street, P. (2006). A Closer Look at Skip-gram Modelling. *Proceedings of the fifth international conference on language resources and evaluation*, pages 1222–1225.

Hamroun, M., Gouider, M. S., and Said, L. B. (2015). Customer Intentions Analysis of Twitter Based on Semantic Patterns. *The 11th International Conference on Semantics, Knowledge and Grids*, pages 2–6.

Hollerit, B. and Kröll, M. (2013). Towards Linking Buyers and Sellers : Detecting Commercial Intent on Twitter. *Proceeding WWW '13 Companion Proceedings of the 22nd International Conference on World Wide Web*, pages 629–632.

Kim, E. (2013). Everything You Wanted to Know about the Kernel Trick. *http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html*.

Kim, H. W., Gupta, S., and Koh, J. (2011). Investigating the intention to purchase digital items in social networking communities: A customer value perspective. *Information and Management*, 48(6):228–234.

Kotler, P. and Zaltman, G. (1971). Social marketing: an approach to planned social change. *The Journal of Marketing*, 35(3):3–12.

Li, C., Weng, J., He, Q., Yao, Y., Datta, A., Sun, A., and Lee, B.-S. (2012). TwiNER: named entity recognition in targeted twitter stream. *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 721–730.

Liddy, E. (2001). Natural Language Processing. *Encyclopedia of Library and Information Science, 2nd Ed.*, pages 70–85.

Lim, K. and Buntine, W. (2014). Twitter Opinion Topic Model: Extracting Product Opinions from Tweets by Leveraging Hashtags and Sentiment Lexicon. *Proceedings of the 23rd ACM International . . .* , pages 1319–1328.

Lin, H.-t. and Lin, C.-j. (2003). A Study on Sigmoid Kernels for SVM and the Training of non-PSD Kernels by SMO-type Methods. pages 1–32.

Liu, X., Zhang, S., Wei, F., and Zhou, M. (2011). Recognizing Named Entities in Tweets. *In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, 1(2008):359–367.

Pang, G., Jiang, S., and Chen, D. (2013). A simple integration of social relationship and text data for identifying potential customers in microblogging. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8346 LNAI(PART 1):397–409.

Pookulangara, S. and Koesler, K. (2011). Cultural influence on consumers' usage of social networks and its' impact on online purchase intentions. *Journal of Retailing and Consumer Services*, 18(4):348–354.

Safavian, S. and Landgrebe, D. (1991). A Survey of Decision Tree Classifier Methodology. 21(3).

Salampasis, M. (2014). Using social media for continuous monitoring and mining of consumer behaviour. *International Journal of . . .* , (Haicta):8–11.

Thackeray, R., Neiger, B. L., Hanson, C. L., and McKenzie, J. F. (2008). Enhancing promotional strategies within social marketing programs: use of Web 2.0 social media. *Health Promotion Practice*, 9(4):338–343.

Wang, X., Wei, F., Liu, X., Zhou, M., and Zhang, M. (2011). Topic sentiment analysis in twitter: a graph-based hashtag sentiment classification approach. *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1031–1040.

Way, O. M. (1996). Knowledge Discovery and Data Mining : Towards a Unifying Framework.

Zampieri, M., Gebre, B. G., and Diwersy, S. (1992). N-gram Language Models and POS Distribution for the Identification of Spanish Varieties.

Zhao, W. X., Jiang, J., He, J., Song, Y., Achananuparp, P., Lim, E.-P., and Li, X. (2011). Topical keyphrase extraction from Twitter. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 379–388.

Appendix I: Weighted Product-Related and Purchase-Related Keywords

Listing 1: *keywords.csv*

```
lamp, 1
light, 1
personal, 0.6
ambiance, 0.7
homework, 0.8
automatic, 0.6
motion, 0.6
buy, 1
purchase, 1
opinion, 0.5
sale, 0.7
for sale, 0.6
offer, 0.6
special offer, 0.6
sale on amazon, 0.5
hot price, 0.5
cheap, 0.5
discount, 0.8
best value, 0.6
amazon deal, 0.8
best deal, 0.8
looking for a deal, 0.7
snap deal, 0.5
want to buy, 0.8
need to buy, 0.8
want to try, 0.8
want to order, 0.8
thinkin to order, 0.8
thinking to order, 0.8
thinkin to try, 0.8
thinking to try, 0.8
thinkin to buy, 0.8
thinking to buy, 0.8
acquire, 0.7
need an advice, 1
work, 0.4
play, 0.4
night, 0.5
dark, 0.7
party, 0.8
firework, 0.7
sensor, 0.6
bought, 1
magic, 0.7
relax, 1
concentrate, 1
```

help, 0.3
room, 0.6
ipad, 0.6
android, 0.7
iphone, 0.7
support, 0.8
custom, 0.4
scene, 0.8
early, 0.6
morning, 0.8
smart, 0.6
future, 0.6
home, 0.8
bulb, 0.6
new, 0.4
question, 0.1
glow, 0.2
app, 0.5
picture, 0.4
color, 0.4
brightness, 0.8
light color, 1
change light, 1
philips hue, 1
energy, 0.4
energy boost, 1
boost energy, 1
connected, 0.7
connected lamp, 1
hue go, 1
travel, 0.6
holiday, 0.6
camping, 0.6
camping light, 1
light schedule, 1
schedule light, 1
warm glow, 1
couch, 0.5
energize, 0.4
energize light, 1
cold, 0.4
chill, 0.6
cozy, 0.8
cozy glow, 1
dimmer, 1
easy to control, 1
control, 0.4
warm light, 1
wellness, 0.7
white light, 1
enlightening, 1
daily routine, 0.7

```
mood, 0.6
turn on, 0.7
turn off, 0.8
automatic light, 1
automatic lamp, 1
motion sensor, 1
personal light, 1
personal lamp, 1
```

Appendix II: Weighted Twitter Accounts of Influencers

Listing 2: *influencers.csv*

```
tweethue, 1
philips, 1
philipspr, 0.7
philipslight, 1
philipslivefrom, 0.6
philips_uk, 0.8
PhilipsNA, 1
PhilipsLightIND, 0.8
PhilipsMenUK, 0.5
philipshuedev, 1
philipshue1, 1
AmbeeApp, 1
HueMenu, 1
HueDisco, 0.8
Huemote, 1
HueParty, 1
huecamera, 1
syncmylights, 0.8
nickisnpdx, 0.7
CancerGeek, 0.6
JohnNosta, 0.7
JoostMaltha, 1
pascalmeijer74, 1
RenateWijma, 1
jeroentas, 0.8
```

Appendix II: Data Package

Listing 3: *Tweet.java*

```
package data;

import twitter4j.JSONException;
import twitter4j.JSONObject;

import java.util.Date;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 20/10/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class Tweet {

    private final long ID;
    private final String TEXT;
    private final Date DATETIME;

    public Tweet(long id, String text, Date datetime) {
        this.ID = id;
        this.TEXT = text;
        this.DATETIME = datetime;
    }

    public JSONObject getJSONObject() {
        JSONObject tweetObj = new JSONObject();
        try {
            tweetObj.put("id", ID);
            tweetObj.put("text", TEXT);
            tweetObj.put("datetime", DATETIME);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return tweetObj;
    }

    public String getText() {
        return TEXT;
    }

    public long getID() {
        return ID;
    }

    public Date getDateTime() { return DATETIME; }

    @Override
    public String toString() {
```

```
        StringBuilder sb = new StringBuilder(TEXT);
        for (int i = 0; i < sb.length(); i++) {
            if (sb.charAt(i) == '@') {
                int start = i;
                int end = sb.indexOf(" ", i) >=0 ? sb.indexOf(" ", i) :
                    sb.length();
                sb.replace(start+1, end, "username");
            }
        }
        return ID + ": " + sb.toString() + "(" + DATETIME + ")";
    }
}
```

Listing 4: *User.java*

```
package data;

import java.util.ArrayList;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 20/10/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class User {

    private final long ID;
    private final String NAME;
    private final String BIO;
    private ArrayList<Tweet> timeline;
    private long idFirstTweet;
    private String sinceDate;
    private double[] unigramFreqs;
    private double[] skipBigramFreqs;
    private int breakpoint;

    private int label;

    public User(long id, String name, String bio) {
        this.ID = id;
        this.NAME = name;
        this.BIO = bio;
        this.timeline = new ArrayList<>();
    }

    public void setSinceDate(String sinceDate) {
        this.sinceDate = sinceDate;
    }

    public void setIdFirstTweet(long id) {
        this.idFirstTweet = id;
    }
```

```
public long getIdFirstTweet() {
    return idFirstTweet;
}

public void fillTimeline(Tweet tweet) {
    timeline.add(tweet);
}

public org.json.simple.JSONObject getJSONObject() {
    org.json.simple.JSONObject userObj = new org.json.simple.
       JSONObject();

    userObj.put("id", ID);
    userObj.put("name", NAME);
    userObj.put("bio", BIO);

    org.json.simple.JSONArray timelineArray = new org.json.simple.
       JSONArray();
    for (Tweet tweet : timeline) {
        timelineArray.add(tweet.getJSONObject());
    }
    userObj.put("timeline", timelineArray);
    return userObj;
}

public long getID() {
    return ID;
}

public String getName() {
    return NAME;
}

public ArrayList<Tweet> getTimeline() {
    return timeline;
}

public int getLabel() {
    return label;
}

public void setLabel(int label) {
    this.label = label;
}

public void setBreakpoint(int breakpoint) { this.breakpoint =
    breakpoint; }

public int getBreakpoint() { return breakpoint; }

public void setUnigramFreqs(double[] unigramFreqs) {
```

```
        this.unigramFreqs = unigramFreqs;
    }

    public void setSkipBigramFreqs(double[] skipBigramFreqs) {
        this.skipBigramFreqs = skipBigramFreqs;
    }

    public double[] getUnigramFreqs() {
        return unigramFreqs;
    }

    public double[] getSkipBigramFreqs() {
        return skipBigramFreqs;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Account: " + NAME);
        sb.append("\n");
        for (Tweet tweet : timeline) {
            sb.append(tweet.toString());
            sb.append("\n");
        }
        return sb.toString();
    }

    @Override
    public boolean equals(Object o) {
        if (o == null) return false;
        if (o.getClass() != this.getClass()) return false;

        User tempUser = (User) o;
        return tempUser.getID() == getID();
    }

}
```

Listing 5: *SkipGram.java*

```
package data;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 07/11/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class SkipGram {

    private final String string1, string2;

    public SkipGram(String s1, String s2) {
```

```
        this.string1 = s1;
        this.string2 = s2;
    }

    @Override
    public boolean equals(Object o) {
        if (o == null) return false;
        if (o.getClass() != this.getClass()) return false;

        SkipGram tempSG = (SkipGram) o;
        return (tempSG.getString1().equals(this.string1) && tempSG.
           getString2().equals(this.string2)) ||
               (tempSG.getString1().equals(this.string2) && tempSG.
                  getString2().equals(this.string1) );
    }

    public String getString1() {
        return string1;
    }
    public String getString2() {
        return string2;
    }

    @Override
    public String toString() {
        return "(" + string1 + ", " + string2 + ")";
    }
}
```

Listing 6: *WeightedPair.java*

```
package data;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 29/11/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class WeightedPair {

    private final String word;
    private final double weight;

    public WeightedPair(String word, double weight) {
        this.word = word;
        this.weight = weight;
    }

    public String getWord() {
        return word;
    }
    public double getWeight() {
```

```
        return weight;
    }
}
```

Appendix III: Util Package

Listing 7: *JSONReader.java*

```
package util;

import data.Tweet;
import data.User;
import main.Main;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 26/10/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class JSONReader {

    private final String path;
    private final JSONParser parser = new JSONParser();

    public JSONReader(String path) {
        this.path = path;
    }

    /**
     * Read dataset from json
     * @return list of users, each with a timeline of tweets
     */
    public ArrayList<User> readJSON() {
        System.out.println(Main.GREEN + "> Reading tweets from JSON
            file." + Main.RESET);
        ArrayList<User> users = new ArrayList<>();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
            HH:mm:ss");
        try {
            Object obj = parser.parse(new FileReader(path));
            JSONObject jsonObject = (JSONObject) obj;
            JSONArray dataArray = (JSONArray) jsonObject.get("data");

            Iterator<JSONObject> iterator = dataArray.iterator();
```

```
            System.out.println(Main.GREEN + "> Parsing user" + Main.
                RESET);
            while (iterator.hasNext()) {

                JSONObject userObject = iterator.next();
                String bio = (String) userObject.get("bio");
                String user_id = (String) userObject.get("id");
                String name = (String) userObject.get("name");
                User u = new User(Long.parseLong(user_id), name, bio);

                String label = (String) userObject.get("label");
                u.setLabel(Integer.parseInt(label));
                int breakpoint = (int) userObject.get("at_tweet");
                u.setBreakpoint(breakpoint);

                JSONArray timeline = (JSONArray) userObject.get("
                    timeline");
                Iterator<JSONObject> iterator_timeline = timeline.
                    iterator();
                while (iterator_timeline.hasNext()) {
                    JSONObject tweetObject = iterator_timeline.next();
                    Date datetime = dateFormat.parse((String) tweetObject
                        .get("datetime"));
                    String tweet_id = (String) tweetObject.get("id");
                    String text = (String) tweetObject.get("text");
                    u.fillTimeline(new Tweet(Long.parseLong(tweet_id),
                        text, datetime));
                }
                users.add(u);
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        } catch (java.text.ParseException e) {
            e.printStackTrace();
        }
        return users;
    }

    public ArrayList<User> readJSON(int experiment) {
        System.out.println(Main.GREEN + "> Reading tweets from JSON
            file." + Main.RESET);
        ArrayList<User> users = new ArrayList<>();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
            HH:mm:ss");
        try {
            Object obj = parser.parse(new FileReader(path));
            JSONObject jsonObject = (JSONObject) obj;
```

```
JSONArray dataArray = (JSONArray) jsonObject.get("data");

Iterator<JSONObject> iterator = dataArray.iterator();
System.out.println(Main.GREEN + "> Parsing user" + Main.
    RESET);
while (iterator.hasNext()) {

    JSONObject userObject = iterator.next();
    String bio = (String) userObject.get("bio");
    String user_id = (String) userObject.get("id");
    String name = (String) userObject.get("name");
    User u = new User(Long.parseLong(user_id), name, bio);

    String label = (String) userObject.get("label");
    u.setLabel(Integer.parseInt(label));
    String breakpoint = (String) userObject.get("at_tweet");
    u.setBreakpoint(Integer.parseInt(breakpoint));

    JSONArray timeline = (JSONArray) userObject.get("
        timeline");
    Iterator<JSONObject> iterator_timeline = timeline.
        iterator();

    if (experiment == 1) {
        while (iterator_timeline.hasNext()) {
            JSONObject tweetObject = iterator_timeline.next()
                ;
            Date datetime = dateFormat.parse((String)
                tweetObject.get("datetime"));
            String tweet_id = (String) tweetObject.get("id");
            String text = (String) tweetObject.get("text");
            u.fillTimeline(new Tweet(Long.parseLong(tweet_id)
                , text, datetime));
        }
    } else {
        int count = 1;
        while (iterator_timeline.hasNext()) {
            JSONObject tweetObject = iterator_timeline.next()
                ;
            if (count >= Integer.parseInt(breakpoint)) {
                Date datetime = dateFormat.parse((String)
                    tweetObject.get("datetime"));
                String tweet_id = (String) tweetObject.get("
                    id");
                String text = (String) tweetObject.get("text"
                    );
                u.fillTimeline(new Tweet(Long.parseLong(
                    tweet_id), text, datetime));
            }
            count++;
        }
    }
```

```
                users.add(u);
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        } catch (java.text.ParseException e) {
            e.printStackTrace();
        }
        return users;
    }
}
```

Listing 8: *PairReader.java*

```
package util;

import data.WeightedPair;
import main.Main;
import org.apache.lucene.search.Weight;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 28/10/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class PairReader {

    public PairReader() {}

    /**
     * Read combinations of keyword and weights.
     * @param path path to files
     * @return list of pairs
     */
    public ArrayList<WeightedPair> readPairsFromFile(String path) {
        ArrayList<WeightedPair> pairs = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(path
            ))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] values = line.split(",");
                double weight = 0.0;
```

```
                try {
                    weight = Double.parseDouble(values[1]);
                } catch(NumberFormatException nfe) {
                    System.out.println(Main.RED + "Invalid csv file: " +
                        path + Main.RESET);
                }
                pairs.add(new WeightedPair(values[0], weight));
            }

        } catch (IOException ignored) {}
        return pairs;
    }

    public boolean canOpenFile(String path) {
        File keywordsFile = new File(path);
        if (!keywordsFile.exists())
            return false;
        return true;
    }

}
```

Listing 9: *SentimentClassifier.java*

```
package util;

import twitter4j.JSONArray;
import twitter4j.JSONException;
import twitter4j.JSONObject;

import com.google.appengine.repackaged.org.apache.http.HttpResponse;
import com.google.appengine.repackaged.org.apache.http.client.
    HttpClient;
import com.google.appengine.repackaged.org.apache.http.client.methods.
    HttpPost;
import com.google.appengine.repackaged.org.apache.http.entity.
    StringEntity;
import com.google.appengine.repackaged.org.apache.http.impl.client.
    DefaultHttpClient;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 27/10/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class SentimentClassifier {

    public SentimentClassifier() {
```

```
}

/**
 * Create a request for getting the sentiment of a tweet
 * @param tweet tweet to classify
 * @param topic topic that is used as extra parameter
 * @return integer representing the sentiment
 */
public int classify(String tweet, String topic) {
    HttpClient httpClient = new DefaultHttpClient();
    HttpResponse response = null;
    int sentiment = -999;
    try {
        HttpPost request = new HttpPost("http://www.sentiment140.com
            /api/bulkClassifyJson?appid=martijn.oele@student.ru.nl");
        StringEntity params =new StringEntity("{\"data\"=[{\"text
            \":\""+tweet+"\", \"topic\":\""+topic+"\"}]}");
        request.addHeader("content-type", "application/json");
        request.addHeader("Accept","application/json");
        request.setEntity(params);
        response = httpClient.execute(request);

        sentiment = parseJSONResult(response);
    }catch (Exception ex) {
    } finally {
        httpClient.getConnectionManager().shutdown();
    }
    return sentiment;
}

public int classify(String tweet) {
    return classify(tweet, "");
}

/**
 * Parse JSON output from sentiment classifier website.
 * @param response json response
 * @return integer representing the sentiment
 */
public int parseJSONResult(HttpResponse response) {
    BufferedReader reader ;
    int sentiment = -999;

    try {
        reader = new BufferedReader(new InputStreamReader(response.
            getEntity().getContent(), "UTF-8"));
        StringBuilder builder = new StringBuilder();
        for (String line; (line = reader.readLine()) != null;) {
            builder.append(line).append("\n");
        }
        if (!builder.toString().equals("")) {
            JSONObject jsonObject = new JSONObject(builder.toString
```

```
                ());
            JSONArray myResponse = jsonObject.getJSONArray("data");

            for(int i=0; i<myResponse.length(); i++){
                sentiment = Integer.parseInt(myResponse.getJSONObject
                    (i).getString("polarity"));
            }
        }

    } catch (IOException e) {
        e.printStackTrace();
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return sentiment;

  }
}
```

Listing 10: *Tagger.java*

```
package util;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import cmu.arktweetnlp.Twokenize;
import cmu.arktweetnlp.impl.Model;
import cmu.arktweetnlp.impl.ModelSentence;
import cmu.arktweetnlp.impl.Sentence;
import cmu.arktweetnlp.impl.features.FeatureExtractor;

/**
 * Tagger object -- wraps up the entire tagger for easy usage from Java
    .
 *
 * To use:
 *
 * (1) call loadModel().
 *
 * (2) call tokenizeAndTag() for every tweet.
 *
 * See main() for example code.
 *
 * (Note RunTagger.java has a more sophisticated runner.
 * This class is intended to be easiest to use in other applications.)
 */
public class Tagger {
    public Model model;
    public FeatureExtractor featureExtractor;
```

```
    /**
     * Loads a model from a file. The tagger should be ready to tag
         after calling this.
     * @param modelFilename
     * @throws IOException
     */
    public void loadModel(String modelFilename) throws IOException {
        model = Model.loadModelFromText(modelFilename);
        featureExtractor = new FeatureExtractor(model, false);
    }

    /**
     * One token and its tag.
     **/
    public static class TaggedToken {
        public String token;
        public String tag;
    }

    /**
     * Run the tokenizer and tagger on one tweet's text.
     **/
    public List<TaggedToken> tokenizeAndTag(String text) {
        if (model == null) throw new RuntimeException("Must loadModel()
             first before tagging anything");
        List<String> tokens = Twokenize.tokenizeRawTweetText(text);

        Sentence sentence = new Sentence();
        sentence.tokens = tokens;
        ModelSentence ms = new ModelSentence(sentence.T());
        featureExtractor.computeFeatures(sentence, ms);
        model.greedyDecode(ms, false);

        ArrayList<TaggedToken> taggedTokens = new ArrayList<TaggedToken
            >();

        for (int t=0; t < sentence.T(); t++) {
            TaggedToken tt = new TaggedToken();
            tt.token = tokens.get(t);
            tt.tag = model.labelVocab.name( ms.labels[t] );
            taggedTokens.add(tt);
        }

        return taggedTokens;
    }

}
```

Listing 11: *Extractor.java*

```
package util;
```

```
import data.SkipGram;
import data.Tweet;
import data.User;
import main.Main;

import java.text.Normalizer;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 05/12/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class Extractor {


    /**
     * Create a list of all unigrams from all timelines in the dataset.
     * @param tagger tagger that is used to assign tags to tokens.
     * @param users dataset
     * @return list of unigrams
     */
    public static ArrayList<String> extractUnigrams(Tagger tagger,
        ArrayList<User> users) {
        ArrayList<String> unigrams = new ArrayList<>();
        System.out.print(Main.GREEN + "> Storing all unigrams" + Main.
            RESET);
        for (User u : users) {
            for (Tweet tweet : u.getTimeline()) {
                List<Tagger.TaggedToken> taggedTokens = tagger.
                    tokenizeAndTag(tweet.getText());
                for (Tagger.TaggedToken taggedToken : taggedTokens) {
                    if (!taggedToken.tag.equals("U")) {
                        String token = Normalizer.normalize(taggedToken.
                            token, Normalizer.Form.NFC);
                        token = token.replaceAll("[^\\p{ASCII}]", "").
                            toLowerCase();

                        if (!unigrams.contains(token))
                            unigrams.add(token);
                    }
                }
            }
        }
        int[] unigramFreqs = new int[unigrams.size()];
        Arrays.fill(unigramFreqs, 0);
        int totalNumberOfTokens = 0;

        for (User u : users) {
            for (Tweet tweet : u.getTimeline()) {
```

```
            List<Tagger.TaggedToken> taggedTokens = tagger.
                tokenizeAndTag(tweet.getText());
            totalNumberOfTokens += taggedTokens.size();
            for (Tagger.TaggedToken taggedToken : taggedTokens) {
                if (!taggedToken.tag.equals("U")) {
                    String token = Normalizer.normalize(taggedToken.
                        token, Normalizer.Form.NFC);
                    token = token.replaceAll("[^\\p{ASCII}]", "").
                        toLowerCase();
                    if (unigrams.contains(token)) {
                        int index = unigrams.indexOf(token);
                        unigramFreqs[index]++;
                    }
                }
            }
        }
    }

    ArrayList<String> filtered_unigrams = new ArrayList<>();
    for (int i = 0; i < unigramFreqs.length; i++) {
        if (unigramFreqs[i] > 3) {
            filtered_unigrams.add(unigrams.get(i));
        }
    }
    System.out.print(" (" + filtered_unigrams.size() + " items) \n"
        );
    return filtered_unigrams;
}

/**
 * Create a list of all skip-bigrams from all timelines in the
     dataset.
 * @param tagger tagger that is used to assign tags to tokens.
 * @param users dataset
 * @return list of skip-bigrams
 */
public static ArrayList<SkipGram> extractSkipBigrams(Tagger tagger,
     ArrayList<User> users) {
    ArrayList<SkipGram> skipBigrams = new ArrayList<>();

    System.out.print(Main.GREEN + "> Storing all skip-bigrams" +
        Main.RESET);
    for (User u : users) {
        for (Tweet tweet : u.getTimeline()) {
            List<Tagger.TaggedToken> taggedTokens = tagger.
                tokenizeAndTag(tweet.getText());
            ArrayList<String> tempTokens = new ArrayList<>();
            for (Tagger.TaggedToken taggedToken : taggedTokens) {
                if (!taggedToken.tag.equals("U") && !taggedToken.tag.
                    equals("@")) {
                    String token = Normalizer.normalize(taggedToken.
                        token, Normalizer.Form.NFC);
```

```
                token = token.replaceAll("[^\\p{ASCII}]", "").
                    toLowerCase();
                tempTokens.add(token);
            }
        }
        for (int i = 0; i < tempTokens.size()-1; i++) {
            for (int j = i + 1; j < tempTokens.size()-1; j++) {
                SkipGram sg = new SkipGram(tempTokens.get(i),
                    tempTokens.get(j));
                if (!skipBigrams.contains(sg))
                    skipBigrams.add(sg);
            }
        }
    }
}

double[] skipBigramFreqs = new double[skipBigrams.size()];
Arrays.fill(skipBigramFreqs, 0.0);
int totalNumberOfTokens = 0;
for (User u : users) {
    for (Tweet tweet : u.getTimeline()) {
        List<Tagger.TaggedToken> taggedTokens = tagger.
            tokenizeAndTag(tweet.getText());
        totalNumberOfTokens += taggedTokens.size();
        ArrayList<String> tempTokens = new ArrayList<>();
        for (Tagger.TaggedToken taggedToken : taggedTokens) {
            if (!taggedToken.tag.equals("U") && !taggedToken.tag.
                equals("@")) {
                String token = Normalizer.normalize(taggedToken.
                    token, Normalizer.Form.NFC);
                token = token.replaceAll("[^\\p{ASCII}]", "").
                    toLowerCase();
                tempTokens.add(token);
            }
        }

        for (int i = 0; i < tempTokens.size(); i++) {
            for (int j = i+1; j < tempTokens.size(); j++) {
                SkipGram sg = new SkipGram(tempTokens.get(i),
                    tempTokens.get(j));
                if (skipBigrams.contains(sg)) {
                    int index = skipBigrams.indexOf(sg);
                    skipBigramFreqs[index]++;
                }
            }
        }
    }
}

ArrayList<SkipGram> filtered_skipbigrams = new ArrayList<>();
for (int i = 0; i < skipBigramFreqs.length; i++) {
    if (skipBigramFreqs[i] > 3) {
```

```
            filtered_skipbigrams.add(skipBigrams.get(i));
        }
    }
    System.out.print(" (" + filtered_skipbigrams.size() + " items)
        \n");
    return filtered_skipbigrams;
}

/**
 * Count all unigrams for a user.
 * @param tagger tagger that is used to assign tags to tokens.
 * @param u user
 * @param unigrams list of all unigrams
 */
public static void countUnigrams(Tagger tagger, User u, ArrayList<
    String> unigrams) {
    System.out.println("\t> Counting unigrams");
    double[] unigramFreqs = new double[unigrams.size()];
    Arrays.fill(unigramFreqs, 0.0);
    int totalNumberOfTokens = 0;

    for (Tweet tweet : u.getTimeline()) {
        List<Tagger.TaggedToken> taggedTokens = tagger.
            tokenizeAndTag(tweet.getText());
        totalNumberOfTokens += taggedTokens.size();
        for (Tagger.TaggedToken taggedToken : taggedTokens) {
            if (!taggedToken.tag.equals("U")) {
                String token = Normalizer.normalize(taggedToken.token
                    , Normalizer.Form.NFC);
                token = token.replaceAll("[^\\p{ASCII}]", "").
                    toLowerCase();
                if (unigrams.contains(token)) {
                    int index = unigrams.indexOf(token);
                    unigramFreqs[index]++;
                }
            }
        }
    }

    for (int i = 0; i < unigramFreqs.length; i++)
        unigramFreqs[i] = unigramFreqs[i]/totalNumberOfTokens;
    u.setUnigramFreqs(unigramFreqs);
}

/**
 * Count all skip-bigrams for a user.
 * @param tagger tagger that is used to assign tags to tokens.
 * @param u user
 * @param skipBigrams list of all skip-bigrams
 */
public static void countSkipBigrams(Tagger tagger, User u,
    ArrayList<SkipGram> skipBigrams) {
```

```
        System.out.println("\t> Counting skip-bigrams");
        double[] skipBigramFreqs = new double[skipBigrams.size()];
        Arrays.fill(skipBigramFreqs, 0);

        int totalNumberOfTokens = 0;

        for (Tweet tweet : u.getTimeline()) {
            List<Tagger.TaggedToken> taggedTokens = tagger.
               tokenizeAndTag(tweet.getText());
            totalNumberOfTokens += taggedTokens.size();
            ArrayList<String> tempTokens = new ArrayList<>();
            for (Tagger.TaggedToken taggedToken : taggedTokens) {
                if (!taggedToken.tag.equals("U") && !taggedToken.tag.
                    equals("@")) {
                    String token = Normalizer.normalize(taggedToken.token
                        , Normalizer.Form.NFC);
                    token = token.replaceAll("[^\\p{ASCII}]", "").
                       toLowerCase();
                    tempTokens.add(token);
                }
            }

            for (int i = 0; i < tempTokens.size(); i++) {
                for (int j = i+1; j < tempTokens.size(); j++) {
                    SkipGram sg = new SkipGram(tempTokens.get(i),
                       tempTokens.get(j));
                    if (skipBigrams.contains(sg)) {
                        int index = skipBigrams.indexOf(sg);
                        skipBigramFreqs[index]++;
                    }
                }
            }
        }

        for (int i = 0; i < skipBigramFreqs.length; i++)
            skipBigramFreqs[i] = skipBigramFreqs[i]/totalNumberOfTokens;
        u.setSkipBigramFreqs(skipBigramFreqs);
    }
}
```

Listing 12: *FeatureReader.java*

```
package util;

import main.Main;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/**
```

```
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 10/01/2017.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class FeatureReader {
    private final String path;
    private ArrayList<ArrayList<Double>> unigramFreqs;
    private ArrayList<ArrayList<Double>> skipBigramFreqs;
    private ArrayList<ArrayList<Double>> abstractFeatures;
    private ArrayList<ArrayList<String>> usersInfo;

    public FeatureReader(String path) {
        this.path = path;

        try {
            unigramFreqs = normalizeDoubles(readFeatures("
                unigram_features.csv"));
            skipBigramFreqs = normalizeDoubles(readFeatures("
                skipbigram_features.csv"));
            abstractFeatures = normalizeDoubles(readFeatures("
                abstract_features.csv", 9));
            usersInfo = readUsersInfo("users_info.csv");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public int getSize() {
        return usersInfo.size();
    }

    private ArrayList<ArrayList<Double>> subList(ArrayList<ArrayList<
        Double>> list, int from, int to) {
        ArrayList<ArrayList<Double>> subList = new ArrayList<>();
        for (int i = from; i < to; i++) {
            subList.add(list.get(i));
        }
        return subList;
    }

    private ArrayList<ArrayList<String>> subListString(ArrayList<
        ArrayList<String>> list, int from, int to) {
        ArrayList<ArrayList<String>> subList = new ArrayList<>();
        for (int i = from; i < to; i++) {
            subList.add(list.get(i));
        }
        return subList;
    }

    public ArrayList<ArrayList<Double>> getUni(int from, int to) {
        if (to < from) {
            ArrayList<ArrayList<Double>> list = subList(unigramFreqs,
```

```
            from, unigramFreqs.size());
        list.addAll(subList(unigramFreqs, 0, to));
        return list;
    }
    return subList(unigramFreqs, from, to);
}
public ArrayList<ArrayList<Double>> getSkip(int from, int to) {
    if (to < from) {
        ArrayList<ArrayList<Double>> list = subList(skipBigramFreqs,
             from, skipBigramFreqs.size());
        list.addAll(subList(skipBigramFreqs, 0, to));
        return list;
    }
    return subList(skipBigramFreqs, from, to);
}
public ArrayList<ArrayList<Double>> getAbstr(int from, int to) {
    System.out.println(from);
    System.out.println(to);
    if (to < from) {
        ArrayList<ArrayList<Double>> list = subList(abstractFeatures
            , from, abstractFeatures.size());
        list.addAll(subList(abstractFeatures, 0, to));
        return list;
    }
    return subList(abstractFeatures, from, to);
}
public ArrayList<ArrayList<String>> getUsers(int from, int to) {
    if (to < from) {
        ArrayList<ArrayList<String>> list = subListString(usersInfo,
             from, usersInfo.size());
        list.addAll(subListString(usersInfo, 0, to));
        return list;
    }
    return subListString(usersInfo, from, to);
}

private ArrayList<ArrayList<Double>> readFeatures(String filename,
    int max) throws IOException {
    System.out.println(Main.GREEN + "> Read features from file " +
       Main.RESET + "[" + filename + "]");
    ArrayList<ArrayList<Double>> features = new ArrayList<>();
    String fileToParse = path + "/features/" + filename;
    BufferedReader fileReader = null;

    final String DELIMITER = ",";

    fileReader = new BufferedReader(new FileReader(fileToParse));
    String line;
    while ((line = fileReader.readLine()) != null) {
        ArrayList<Double> subList = new ArrayList<>();
        String[] tokens = line.split(DELIMITER);
        for (int n = 0; n < max; n++)
```

```
            subList.add(Double.parseDouble(tokens[n]));
        features.add(subList);
    }
    fileReader.close();
    return features;
}

private ArrayList<ArrayList<Double>> readFeatures(String filename)
    throws IOException {
    System.out.println(Main.GREEN + "> Read features from file " +
        Main.RESET + "[" + filename + "]");
    ArrayList<ArrayList<Double>> features = new ArrayList<>();
    String fileToParse = path + "/features/" + filename;
    BufferedReader fileReader = null;

    final String DELIMITER = ",";

    fileReader = new BufferedReader(new FileReader(fileToParse));
    String line;
    while ((line = fileReader.readLine()) != null) {
        ArrayList<Double> subList = new ArrayList<>();
        String[] tokens = line.split(DELIMITER);
        for (String token : tokens)
            subList.add(Double.parseDouble(token));
        features.add(subList);

    }
    fileReader.close();
    return features;
}

private ArrayList<ArrayList<String>> readUsersInfo(String filename)
     throws IOException {
    ArrayList<ArrayList<String>> usersInfo = new ArrayList<>();
    String fileToParse = path + "/features/" + filename;
    BufferedReader fileReader = null;

    final String DELIMITER = ",";

    fileReader = new BufferedReader(new FileReader(fileToParse));
    String line;
    while ((line = fileReader.readLine()) != null)
    {
        ArrayList<String> subList = new ArrayList<>();
        String[] tokens = line.split(DELIMITER);
        for(String token : tokens)
            subList.add(token);
        usersInfo.add(subList);
    }
    fileReader.close();
    return usersInfo;
}
```

```
    private ArrayList<ArrayList<Double>> normalizeDoubles(ArrayList<
        ArrayList<Double>> data) {
        System.out.println(Main.GREEN + "> Normalizing data" + Main.
            RESET);
        ArrayList<ArrayList<Double>> norm_data = new ArrayList<>();
        double dataHigh = 0, dataLow = 9999;

        for (ArrayList<Double> features : data) {
            for (Double feature : features) {
                if (feature > dataHigh)
                    dataHigh = feature;
                if (feature < dataLow)
                    dataLow = feature;
            }
            ArrayList<Double> norm_features = new ArrayList<>();
            for (Double feature : features) {
                norm_features.add(scale(feature, -1, 1, dataLow,
                    dataHigh));
            }
            norm_data.add(norm_features);
        }
        return norm_data;
    }

    private double scale(double unscaled, double min, double max,
        double dataLow, double dataHigh) {
        return (max - min) * (unscaled - dataLow) / (dataHigh - dataLow
            ) + min;
    }
}
```

Appendix IV: TweetManager Package

Listing 13: *TweetManager.java*

```
package tweetmanager;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import data.User;
import main.Main;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.HttpClientBuilder;
import org.json.JSONObject;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
  import twitter4j.*;

/**
 * Class to getting tweets based on username and optional time
    constraints
 *
 * @author Jefferson Henrique
 */
public class TweetManager {

  /**
   * @param username A specific username (without @)
   * @param since Lower bound date (yyyy-mm-dd)
   * @param until Upper bound date (yyyy-mm-dd)
   * @param scrollCursor (Parameter used by Twitter to do pagination of
       results)
   * @return JSON response used by Twitter to build its results
   * @throws Exception
   */
  private static String getURLResponse(String username, String since,
     String until, String querySearch, String lang, String scrollCursor
     ) throws Exception {
    String appendQuery = "";
    if (username != null)
      appendQuery += "from:"+username;
    if (since != null)
      appendQuery += " since:"+since;
    if (until != null)
      appendQuery += " until:"+until;
```

```
  if (querySearch != null)
    appendQuery += " "+querySearch;
  if (lang != null)
    appendQuery += " lang:"+lang;

  String url = String.format("https://twitter.com/i/search/timeline?f
     =realtime&q=%s&src=typd&max_position=%s", URLEncoder.encode(
     appendQuery, "UTF-8"), scrollCursor);

  HttpClient client = HttpClientBuilder.create().build();
     HttpGet request = new HttpGet(url);
     HttpResponse response = client.execute(request);
     BufferedReader rd = new BufferedReader(new InputStreamReader(
        response.getEntity().getContent()));

     StringBuffer result = new StringBuffer();
     String line;
     while ((line = rd.readLine()) != null) {
        result.append(line);
     }

     return result.toString();
}

public static List<User> getUsers(TwitterCriteria criteria) {
     List<User> results = new ArrayList<>();

     try {
        String refreshCursor = null;
        outerLace: while (true) {
     JSONObject json = new JSONObject(getURLResponse(criteria.
        getUsername(), criteria.getSince(), criteria.getUntil(),
        criteria.getQuerySearch(), criteria.getLang(), refreshCursor
        ));
            refreshCursor = json.getString("min_position");
            Document doc = Jsoup.parse((String) json.get("items_html
               "));
            Elements tweets = doc.select("div.js-stream-tweet");

            if (tweets.size() == 0)
               break;

            for (Element tweet : tweets) {
       String username = tweet.select("span.username.js-action-
          profile-name b").text();
               String id = tweet.attr("data-tweet-id");
               long dateMs = Long.valueOf(tweet.select("small.time
                  span.js-short-timestamp").attr("data-time-ms"));
               Date date = new Date(dateMs);
       Twitter twitter = TwitterFactory.getSingleton();
               twitter4j.User twUser = twitter.showUser(username);
```

```
                User u = new User(twUser.getId(), twUser.
                    getScreenName(), twUser.getDescription());
                u.setIdFirstTweet(Long.parseLong(id));
        u.setSinceDate(date.toString());
        if (!results.contains(u))
                    results.add(u);

                if (criteria.getMaxTweets() > 0 && results.size() >=
                    criteria.getMaxTweets())
                    break outerLace;
        System.out.print(".");
      }
          }
      } catch (Exception e) {
          e.printStackTrace();
      }
  System.out.print("\n");
  return results;

  }

public static List<Status> getNTweets(long user_id, int n, long
    first_id) {
  Twitter twitter = new TwitterFactory().getInstance();
  List statuses = new ArrayList();
  int pageno = 1;

  while (true) {
    try {
      int size = statuses.size();
      long sinceId = new Double(first_id * 0.995).longValue();
      Paging page = new Paging(pageno++, n).sinceId(sinceId);
      statuses.addAll(twitter.getUserTimeline(user_id, page));
      if (statuses.size() == size)
        break;
    }
    catch(TwitterException e) {
      System.out.println(Main.RED + "Rate limit exceeded. Program
          will sleep until limit has been reset." + Main.RESET);
      try {
        Thread.sleep(900 * 1000);
      } catch(InterruptedException ex) {
        Thread.currentThread().interrupt();
      }
    }
  }

  Status lastStatus = (Status) statuses.get(0);
  Date startDate = lastStatus.getCreatedAt();
  for (Object o : statuses) {
    Status status = (Status) o;
    if (status.getId() == first_id)
```

```
        startDate = status.getCreatedAt();
    }

    ArrayList<Status> result = new ArrayList<>();
    int tweetsBeforeMention = 0;
    for (Object o : statuses) {
      Status status = (Status) o;
      Date statusDate = status.getCreatedAt();
      int daysBetween = daysBetween(startDate, statusDate);
      if ( (daysBetween <= 61) && (daysBetween >= 0))
        result.add(status);
      else if (daysBetween >= -10 && tweetsBeforeMention <= 20) {
        result.add(status);
        tweetsBeforeMention++;
      }
    }
    if (result.size() >= n)
      return result.subList(result.size()-n, result.size());
    else if (result.size() >= 40)
      return result;
    else
      return null;
  }

  public static int daysBetween(Date d1, Date d2){
    if (d2.getTime() >= d1.getTime())
      return (int)( (d2.getTime() - d1.getTime()) / (1000 * 60 * 60 *
          24));
    else
      return (int)( -1 * (d1.getTime() - d2.getTime()) / (1000 * 60 *
          60 * 24));
  }

}
```

Listing 14: *TwitterCriteria.java*

```
package tweetmanager;

/**
 * A class to guide how the tweets must be searched on {@link
    TweetManager}
 *
 * @author Jefferson Henrique
 *
 */
public class TwitterCriteria {

  private String username;
  private String since;
  private String until;
  private String querySearch;
```

```
private String lang;
private int maxTweets;

private TwitterCriteria() {
}

public static TwitterCriteria create() {
  return new TwitterCriteria();
}

/**
 * @param username (whitout @) Username of a specific twitter account
 *
 * @return Current {@link TwitterCriteria}
 */
public TwitterCriteria setUsername(String username) {
  this.username = username;
  return this;
}

/**
 * @param since The lower bound date (yyyy-mm-aa)
 *
 * @return Current {@link TwitterCriteria}
 */
public TwitterCriteria setSince(String since) {
  this.since = since;
  return this;
}

/**
 * @param until The upper bound date (yyyy-mm-aa)
 *
 * @return Current {@link TwitterCriteria}
 */
public TwitterCriteria setUntil(String until) {
  this.until = until;
  return this;
}

/**
 * @param querySearch A query text to be matched
 *
 * @return Current TwitterCriteria
 */
public TwitterCriteria setQuerySearch(String querySearch) {
  this.querySearch = querySearch;
  return this;
}

/**
 * @param maxTweets The maximum number of tweets to retrieve
```

```
     *
     * @return Current {@link TwitterCriteria}
     */
    public TwitterCriteria setMaxTweets(int maxTweets) {
      this.maxTweets = maxTweets;
      return this;
    }

    public TwitterCriteria setLang(String lang) {
      this.lang = lang;
      return this;
    }

    String getUsername() {
      return username;
    }

    String getSince() {
      return since;
    }

    String getUntil() {
      return until;
    }

    String getQuerySearch() {
      return querySearch;
    }

    int getMaxTweets() {
      return maxTweets;
    }

    String getLang() { return lang; }

}
```

Appendix V: Algorithm Package

Listing 15: *Main.java*

```
package main;

import algorithm.FeatureExtractor;
import algorithm.Tester;
import algorithm.Trainer;
import algorithm.TweetCollector;
import data.WeightedPair;
import libsvm.svm;
import libsvm.svm_model;
import org.apache.http.impl.client.FutureRequestExecutionMetrics;
import util.FeatureReader;
import util.PairReader;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 20/10/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class Main {

    public static final String RED = "\u001B[31m";
    public static final String GREEN = "\u001B[32m";
    public static final String RESET = "\u001B[0m";
    private ArrayList<WeightedPair> keywords;
    private ArrayList<WeightedPair> influencers;
    private svm_model uniModel, skipModel, finalModel;
    private int[] parameter_options;

    // ----------------------------------------------------------------
    private String WORKING_DIRECTORY = "/Users/Martijn/Documents/Thesis
        /";
    private static ArrayList<String> SELECTION_KEYWORDS = new ArrayList
        <>();
    private static final int NUMBER_OF_USERS = 300;
    private static final int MAX_NUMBER_OF_TWEETS_PER_USER = 80;
    private static final String UNTIL_DATE = "2016-10-01";
    // ----------------------------------------------------------------

    public Main() {

    }

    private void start() {
```

```
        System.out.println("Identifying Potential Customers in Tweets (
            Martijn Oele)");
        readFilePath();
        SELECTION_KEYWORDS.add("\"personal light\"");
        SELECTION_KEYWORDS.add("\"personal lamp\"");
        SELECTION_KEYWORDS.add("\"ambiance light\"");
        SELECTION_KEYWORDS.add("\"automatic light\"");
        SELECTION_KEYWORDS.add("\"homework ambiance\"");
        if (haveReadFiles()) {
            int input = 1;
            while (input != 0) {
                showMenu();
                input = readInput();
                processInput(input);
            }
        }
    }

    private void showMenu() {
        System.out.println("------------------------");
        System.out.println("       MENU");
        System.out.println("------------------------");
        System.out.println(" 1)\tCrawl tweets");
        System.out.println(" 2)\tExtract features experiment 1");
        System.out.println(" 3)\tExtract features experiment 2");
        System.out.println(" 4)\tTrain & Test experiments");
        System.out.println(" 0)\tExit");
        System.out.println("------------------------");
    }

    private int readInput() {
        int input;
        while(true) {
            System.out.print("Enter choice: ");
            try {
                input = new Scanner(System.in).nextInt();
                if(input == 1 || input == 2 || input == 3 || input == 4
                    || input == 0)
                    return input;
            }
            catch (Exception e) {
                System.out.println(RED + "> Please enter a number." +
                    RESET);
            }
        }
    }

    private void readFilePath() {
        System.out.print("Enter path to directory <Leave empty for
            default path>: ");
        String input = new Scanner(System.in).nextLine();
        if (!input.equals(""))
```

```
        this.WORKING_DIRECTORY = input;
}

private boolean haveReadFiles() {
    PairReader pr = new PairReader();
    if (pr.canOpenFile(this.WORKING_DIRECTORY + "keywords.txt")) {
        System.out.println(GREEN + "Reading keywords from file" +
           RESET);
        this.keywords = pr.readPairsFromFile(this.WORKING_DIRECTORY
           + "keywords.txt");
    } else {
        System.out.println(RED + "Can not read keywords from file" +
            RESET);
        return false;
    }
    if (pr.canOpenFile(this.WORKING_DIRECTORY + "influencers.txt"))
        {
        System.out.println(GREEN + "Reading influencers from file" +
            RESET);
        this.influencers = pr.readPairsFromFile(this.
           WORKING_DIRECTORY + "influencers.txt");
    } else {
        System.out.println(RED + "Can not read influencers from file
           " + RESET);
        return false;
    }

    return true;
}

private void processInput(int input) {
    switch(input) {
        case 1:
            collect();
            break;
        case 2:
            extract(1);
            break;
        case 3:
            extract(2);
            break;
        case 4:
            train();
            break;
        default: break;
    }
    System.out.println("");
}

private void collect() {
    TweetCollector tc = new TweetCollector(SELECTION_KEYWORDS);
    tc.collectUsers(NUMBER_OF_USERS, UNTIL_DATE);
```

```
        tc.collectTweets(MAX_NUMBER_OF_TWEETS_PER_USER);
        tc.storeTweets(WORKING_DIRECTORY);
    }

    private void extract(int experiment) {
        FeatureExtractor fe = new FeatureExtractor(WORKING_DIRECTORY,
            keywords, influencers);
        fe.extract(experiment);
    }

    private void train() {
        System.out.println(Main.GREEN + "> Training phase " + Main.
            RESET);
        FeatureReader fr = new FeatureReader(WORKING_DIRECTORY);
        int n = fr.getSize();
        Trainer[] trainers = new Trainer[10];
        Thread[] threads = new Thread[10];
        for (int i = 0; i < 10; i++) {
            int i0 = (int) ((i * 0.1 + 0.0)%1.0 * n);
            int i1 = (int) ((i * 0.1 + 0.8)%1.0 * n);
            int i1end = i1;
            int i2 = (int) ((i * 0.1 + 0.9)%1.0 * n);
            int i2end = i2;
            if (i1 == 0)
                i1end = n;
            if (i2 == 0)
                i2end = n;
            Trainer trainer = new Trainer(fr.getUsers(i0, i1end), fr.
                getUsers(i1, i2end));
            trainer.setData(fr.getUni(i0, i1end), fr.getUni(i1, i2end),
                fr.getSkip(i0, i1end), fr.getSkip(i1, i2end), fr.getAbstr
                (i0, i1end), fr.getAbstr(i1, i2end));
            Thread thread = new Thread(trainer);
            thread.start();
            trainers[i] = trainer;
            threads[i] = thread;
        }
        try {
            for (int i = 0; i < 10; i++)
                threads[i].join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        trainers[0].showResults(0);

        System.out.println(Main.GREEN + "> Testing phase" + Main.RESET)
            ;


        ArrayList<Double> labels = new ArrayList<>();
        ArrayList<Double> predictions = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
```

```
    int i2 = (int) ((i * 0.1 + 0.9)%1.0 * n);
    int i3 = (int) ((i * 0.1 + 1.0)%1.0 * n);
    if (i3 == 0)
        i3 = n;
    ArrayList<ArrayList<String>> users = fr.getUsers(i2, i3);
    ArrayList<ArrayList<Double>> uniTest = fr.getUni(i2, i3);
    ArrayList<ArrayList<Double>> skipTest = fr.getSkip(i2, i3);
    ArrayList<ArrayList<Double>> abstrTest = fr.getAbstr(i2, i3)
        ;
    for (int t = 0; t < users.size(); t++) {
        abstrTest.get(i).add(trainers[i].applyUniModel(uniTest.
            get(i)));
        abstrTest.get(i).add(trainers[i].applySkipModel(skipTest
            .get(i)));

        labels.add(Double.parseDouble(users.get(i).get(2)));
        predictions.add(trainers[i].applyAbstractModel(abstrTest
            .get(i)));
    }
}


System.out.println(Main.GREEN + "\n> Results of test " + Main.
    RESET);
System.out.println("\tThreshold\tPrecision\tRecall");

int finaltp = -1, finaltn = -1, finalfp = -1, finalfn = -1;
for (double threshold = 0; threshold < 1; threshold += 0.1) {
    int tp = 0, tn = 0, fp = 0, fn = 0;
    for (int i = 0; i < labels.size(); i++) {
        int label = labels.get(i).intValue();
        int pred = predictions.get(i).compareTo(threshold) > 0 ?
             1 : 0;
        if (label == pred) {
            if (label == 1)
                tp++;
            else
                tn++;
        } else {
            if (label == 1)
                fn++;
            else
                fp++;
        }
        if (threshold == 0.2) {
            finaltp = tp;
            finaltn = tn;
            finalfp = fp;
            finalfn = fn;
        }

    }
```

```
            printMetrics(tp, tn, fp, fn, threshold);
        }
        System.out.println(Main.GREEN + "> Confusion matrix" + Main.
            RESET);
        System.out.println("\t\t\t Predicted");
        System.out.println("\t\t\t | 1 | 0 ");
        System.out.println("\t\t-------------");
        System.out.println("\tTrue 1 | " + finaltp + " | " + finalfn);
        System.out.println("\tclass 0 | " + finalfp + " | " + finaltn);
    }

    private void printMetrics(int tp, int tn, int fp, int fn, double
        threshold) {
        double precision = precision(tp, fp);
        double recall = recall(tp, fn);
        Double f1_new = 2 * ((precision * recall) / (precision + recall
            ));
        f1_new = f1_new.isNaN() ? 0.0 : f1_new;
        System.out.println("\t" + String.format("%.01f", threshold) + "
            \t\t" + String.format("%.02f", precision) + "\t\t" + String.
            format("%.02f", recall));
    }

    private double precision(int tp, int fp) {
        double precision = tp / (double)(tp+fp);
        return Double.isNaN(precision) ? 0.0 : precision;
    }

    private double recall(int tp, int fn) {
        double recall = tp / (double)(tp+fn);
        return Double.isNaN(recall) ? 0.0 : recall;
    }

    public static void main(String[] args) {
        Main main = new Main();
        main.start();
    }
}
```

Listing 16: *TweetCollector.java*

```
package algorithm;

import data.Tweet;
import data.User;
import main.Main;
import tweetmanager.TweetManager;
import tweetmanager.TwitterCriteria;
import twitter4j.Status;

import java.io.*;
import java.util.ArrayList;
```

```
import java.util.Collection;
import java.util.Collections;
import java.util.List;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 20/10/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class TweetCollector {

    private final ArrayList<String> keywords;
    private List<data.User> users;

    public TweetCollector(ArrayList<String> keywords) {
        this.keywords = keywords;
    }

    /**
     * Collect a number of users that have used one of the keywords
         before a certain date.
     * @param n number of users
     * @param untilDate date to check if a users has used a keyword
     */
    public void collectUsers(int n, String untilDate) {
        System.out.println(Main.GREEN + "> Collecting +-" + n + " users
            " + Main.RESET);
        users = new ArrayList<>();

        for (String keyword : keywords) {
            TwitterCriteria criteria = TwitterCriteria.create()
                    .setUntil(untilDate)
                    .setQuerySearch(keyword)
                    .setMaxTweets(n/(keywords.size()))
                    .setLang("en");

            for (User u : TweetManager.getUsers(criteria)) {
                if (!users.contains(u))
                    users.add(u);
            }
        }
    }

    /**
     * Collect a number of tweets for every user. If users has too few
         tweets, remove him from the set.
     * @param n number of tweets needed
     */
    public void collectTweets(int n) {
        ArrayList<Integer> indicesToRemove = new ArrayList<>();
        int index = 0;
        for (data.User user : users) {
```

```
            System.out.println(Main.GREEN + "> Collecting tweets for
                user " + user.getID() + " (" + (index+1) + " of " + users
                .size() + ")" + Main.RESET);
            List<Status> tweets = TweetManager.getNTweets(user.getID(),
                n, user.getIdFirstTweet());
            if (tweets != null && !user.getName().contains("tweethue"))
                {
                for (Status tweet : tweets) {
                    user.fillTimeline(new Tweet(tweet.getId(), tweet.
                        getText(), tweet.getCreatedAt()));
                }
                System.out.println("\t Collected " + user.getTimeline().
                    size() + " tweets");
            } else {
                indicesToRemove.add(users.indexOf(user));
                System.out.println("\t User has too few tweets, will be
                    removed from dataset.");
            }
            index++;
        }
        for (int i = indicesToRemove.size()-1; i >= 0; i--) {
            users.remove((int) indicesToRemove.get(i));
        }
    }

    /**
     * Store tweets in file
     * @param path path to file
     */
    public void storeTweets(String path) {
        Collections.shuffle(users);
        int nr_users = users.size();

        System.out.println("\n* Collected: " + nr_users + " users");

        org.json.simple.JSONArray userArray = new org.json.simple.
            JSONArray();
        for (int i = 0; i < nr_users; i++) {
            userArray.add(users.get(i).getJSONObject());
        }
        org.json.simple.JSONObject dataObj = new org.json.simple.
            JSONObject();
        dataObj.put("data", userArray);

        try {
            FileWriter file = new FileWriter(path + "tweets.json");
            file.write(dataObj.toJSONString());
            file.flush();
            file.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

Listing 17: *FeatureExtractor.java*

```
package algorithm;

import data.SkipGram;
import data.Tweet;
import data.User;
import data.WeightedPair;
import main.Main;
import tweetmanager.TweetManager;
import twitter4j.Twitter;
import twitter4j.TwitterException;
import twitter4j.TwitterFactory;
import util.Extractor;
import util.JSONReader;
import util.SentimentClassifier;
import util.Tagger;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.*;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 26/10/2016.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class FeatureExtractor {
    private final Twitter twitter = new TwitterFactory().getInstance();
    private final Tagger tagger = new Tagger();
    private final SentimentClassifier sentClassifier = new
        SentimentClassifier();
    private final String path;
    private final ArrayList<WeightedPair> keywords;
    private final ArrayList<WeightedPair> influencers;

    public FeatureExtractor(String path, ArrayList<WeightedPair>
        keywords, ArrayList<WeightedPair> influencers) {
        this.path = path;
        this.keywords = keywords;
        this.influencers = influencers;

        try {
            tagger.loadModel("/cmu/arktweetnlp/model.20120919");
        } catch (IOException e) {
            System.out.println(Main.RED + "* Can not load tagger model"
                + Main.RESET);
```

```
    }
}

/**
 * Make a list of unigrams and skip-bigrams, count them on a
    separate thread and
 * extract all features. For every user, create a line with
    features in csv files.
 */
public void extract(int experiment) {
    try {
        PrintWriter pw_1 = new PrintWriter(new File(path + "/
            features/abstract_features.csv"));
        PrintWriter pw_2 = new PrintWriter(new File(path + "/
            features/unigram_features.csv"));
        PrintWriter pw_3 = new PrintWriter(new File(path + "/
            features/skipbigram_features.csv"));
        PrintWriter pw_4 = new PrintWriter(new File(path + "/
            features/users_info.csv"));

        ArrayList<User> data = readData(experiment);
        ArrayList<String> unigrams = Extractor.extractUnigrams(
            tagger, data);
        ArrayList<SkipGram> skipBigrams = Extractor.
            extractSkipBigrams(tagger, data);

        for (User user : data) {
            StringBuilder abstractFeaturesString = new StringBuilder
                ();
            StringBuilder unigramFeaturesString = new StringBuilder
                ();
            StringBuilder skipbigramFeaturesString = new
                StringBuilder();
            StringBuilder usersString = new StringBuilder();

            System.out.println(Main.GREEN + "> Extracting all
                features for user " + (user.getID()+1) + " (" + data.
                indexOf(user) + "/" + data.size() + ")" + Main.RESET)
                ;

            Thread unigramThread = new Thread(() -> {
                Extractor.countUnigrams(tagger, user, unigrams);
            });
            Thread skipBigramThread = new Thread(() -> {
                Extractor.countSkipBigrams(tagger, user, skipBigrams)
                    ;
            });
            unigramThread.start();
            skipBigramThread.start();

            try {
                unigramThread.join();
```

```
            skipBigramThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        double[] features = extractAbstractFeatures(user);

        double[] unigramFreqs = user.getUnigramFreqs();
        double[] skipBigramFreqs = user.getSkipBigramFreqs();

        user.setUnigramFreqs(new double[0]);
        user.setSkipBigramFreqs(new double[0]);

        for (int i = 0; i < unigramFreqs.length; i++) {
            unigramFeaturesString.append(unigramFreqs[i] + ", ");
        }
        unigramFeaturesString.deleteCharAt(unigramFeaturesString
            .length()-1);
        unigramFeaturesString.deleteCharAt(unigramFeaturesString
            .length()-1);
        unigramFeaturesString.append("\n");

        for (int i = 0; i < skipBigramFreqs.length; i++) {
            skipbigramFeaturesString.append(skipBigramFreqs[i] +
                ", ");
        }
        skipbigramFeaturesString.deleteCharAt(
            skipbigramFeaturesString.length()-1);
        skipbigramFeaturesString.deleteCharAt(
            skipbigramFeaturesString.length()-1);
        skipbigramFeaturesString.append("\n");

        for (int i = 0; i < features.length; i++) {
            abstractFeaturesString.append(features[i] + ", ");
        }
        abstractFeaturesString.deleteCharAt(
            abstractFeaturesString.length()-1);
        abstractFeaturesString.deleteCharAt(
            abstractFeaturesString.length()-1);
        abstractFeaturesString.append("\n");

        usersString.append(user.getID() + ", " + user.getName()
            + ", " + user.getLabel() + ", " + user.getBreakpoint
            () + "\n");
        pw_1.write(abstractFeaturesString.toString());
        pw_2.write(unigramFeaturesString.toString());
        pw_3.write(skipbigramFeaturesString.toString());
        pw_4.write(usersString.toString());
    }

    System.out.println(Main.GREEN + "> Storing all features to
        csv files." + Main.RESET);
```

```
            pw_1.close();
            pw_2.close();
            pw_3.close();
            pw_4.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }


}

/**
 * Extract the abstract features for a user
 * @param u user
 * @return an array with 9 abstract features
 */
private double[] extractAbstractFeatures(User u) {
    System.out.println("\t> Extracting abstract features");

    double[] abstractFeatures = new double[9];
    HashMap<Tweet, Double> relevantTweets = new HashMap<>();

    for (Tweet tweet : u.getTimeline()) {
        double relevance_score = 0.0;
        for (WeightedPair keyword : keywords) {
            if (tweet.getText().contains(keyword.getWord()))
                relevance_score += 10 * keyword.getWeight();
        }
        if (relevance_score > 0.0)
            relevantTweets.put(tweet, relevance_score);
    }

    double max_relevance_score = 0.0;
    double total_relevance_score = 0.0;
    double total_sentiment = 0.0;
    Date date_max_relevance_score = new Date();
    int sentiment_max_relevance = 0;

    Set set = relevantTweets.entrySet();
    Iterator iterator = set.iterator();

    double relevance_score_last_relevant_tweet = 0.0;
    Tweet last_relevant_tweet = u.getTimeline().get(0);
    while(iterator.hasNext()) {
        Map.Entry entry = (Map.Entry)iterator.next();
        Tweet tweet = (Tweet) entry.getKey();

        if ((Double) entry.getValue() > max_relevance_score) {
            max_relevance_score = (Double) entry.getValue();
            date_max_relevance_score = tweet.getDateTime();
            sentiment_max_relevance = sentClassifier.classify(tweet.
                getText());
```

```
            }

            total_relevance_score += (Double) entry.getValue();
            total_sentiment += sentClassifier.classify(tweet.getText());
            relevance_score_last_relevant_tweet = (Double) entry.
                getValue();
            last_relevant_tweet = tweet;
        }

        double following_influencing_accounts = 0.0;
        for (WeightedPair influencer : influencers) {
            try {
                boolean isFollowing = twitter.showFriendship( u.getName
                    (), influencer.getWord() ).isSourceFollowingTarget();
                if (isFollowing)
                    following_influencing_accounts += 10 * influencer.
                        getWeight();
            } catch (TwitterException e) {
                if (e.getErrorCode() != 429) {
                    following_influencing_accounts = 0;
                    break;
                } else {
                    System.out.println(Main.RED + "Rate limit exceeded.
                        Program will sleep until limit has been reset." +
                         Main.RESET);
                    try {
                        Thread.sleep(900 * 1000);
                    } catch (InterruptedException ex) {
                        Thread.currentThread().interrupt();
                    }
                }
            }
        }

        abstractFeatures[0] = max_relevance_score;
        abstractFeatures[1] = (double) TweetManager.daysBetween(
            date_max_relevance_score, u.getTimeline().get(0).getDateTime
            ());
        abstractFeatures[2] = (double) sentiment_max_relevance;
        abstractFeatures[3] = total_relevance_score/set.size();
        abstractFeatures[4] = total_sentiment/set.size();
        abstractFeatures[5] = relevance_score_last_relevant_tweet;
        abstractFeatures[6] = (double) TweetManager.daysBetween(
            last_relevant_tweet.getDateTime(), u.getTimeline().get(0).
            getDateTime());
        abstractFeatures[7] = (double) sentClassifier.classify(
            last_relevant_tweet.getText());
        abstractFeatures[8] = following_influencing_accounts;
        return abstractFeatures;
    }

    private ArrayList<User> readData(int experiment) {
```

```
        ArrayList<User> data = new ArrayList<>();
        JSONReader jsonReader = new JSONReader(path + "/dataset.json");
        data.addAll(jsonReader.readJSON(experiment));

        return data;
    }

}
```

Listing 18: *Trainer.java*

```
package algorithm;

import libsvm.*;
import libsvm.svm;
import main.Main;

import java.io.IOException;
import java.util.ArrayList;

import static libsvm.svm_parameter.C_SVC;

/**
 * Created by Martijn Oele (martijn.oele@student.ru.nl) on 05/01/2017.
 * Bachelor's Thesis Artificial Intelligence
 * Radboud University Nijmegen
 */
public class Trainer implements Runnable {

    private double[] train_labels;
    private double[] tune_labels;
    private double[] labels;
    private double[] f1s = new double[6];
    private double[] cs = new double[6];
    private double[] gammas = new double[6];
    private svm_model uniModel;
    private svm_model skipModel;
    private svm_model abstractModel;
    private svm_problem problem_train_uni;
    private svm_problem problem_tune_uni;
    private svm_problem problem_train_skip;
    private svm_problem problem_tune_skip;
    private svm_problem problem_train_abstr;
    private svm_problem problem_tune_abstr;
    private svm_problem problem_uni;
    private svm_problem problem_skip;
    private svm_problem problem_abstr;
    private int index1, index2, index3;
    private ArrayList<ArrayList<Double>> train_abstr;
    private ArrayList<ArrayList<Double>> tune_abstr;

    public Trainer(ArrayList<ArrayList<String>> train_users, ArrayList<
```

```
    ArrayList<String>> tune_users) {
    train_labels = new double[train_users.size()];
    tune_labels = new double[tune_users.size()];
    labels = new double[train_users.size() + tune_users.size()];
    for (int i = 0; i < train_labels.length; i++) {
        train_labels[i] = Double.parseDouble(train_users.get(i).get
            (2));
        labels[i] = Double.parseDouble(train_users.get(i).get(2));
    }
    for (int i = 0; i < tune_labels.length; i++) {
        tune_labels[i] = Double.parseDouble(tune_users.get(i).get(2)
            );
        labels[train_labels.length+i] = Double.parseDouble(
            tune_users.get(i).get(2));
    }
}

/**
 * This function creates an svm_problem of a dataset
 * @param data features
 * @param labels true labels
 * @return svm_problem with nodes and labels
 */
private svm_problem createProblem(ArrayList<ArrayList<Double>> data
    , double[] labels) {
    int l = data.size();
    int n = data.get(0).size();
    svm_node[][] x = new svm_node[l][n];

    for (int i = 0; i < l; i++) {
        ArrayList<Double> subList = data.get(i);
        for (int j = 0; j < n; j++) {
            svm_node node = new svm_node();
            node.index = j;
            node.value = subList.get(j);
            x[i][j] = node;
        }
    }
    svm_problem problem = new svm_problem();
    problem.l = l;
    problem.x = x;
    problem.y = labels;
    return problem;
}

/**
 * Create a problem of a training and tune set
 * @param data1 training set
 * @param data2 tune set
 * @param labels true labels
 * @return svm_problem with nodes and labels
 */
```

```
private svm_problem createProblem(ArrayList<ArrayList<Double>>
    data1, ArrayList<ArrayList<Double>> data2, double[] labels) {
    ArrayList<ArrayList<Double>> new_data = data1;
    new_data.addAll(data2);
    return createProblem(new_data, labels);
}

public void setData(ArrayList<ArrayList<Double>> train_uni,
    ArrayList<ArrayList<Double>> tune_uni, ArrayList<ArrayList<
    Double>> train_skip, ArrayList<ArrayList<Double>> tune_skip,
    ArrayList<ArrayList<Double>> train_abstr, ArrayList<ArrayList<
    Double>> tune_abstr) {
    this.problem_train_uni = createProblem(train_uni, train_labels)
        ;
    this.problem_tune_uni = createProblem(tune_uni, tune_labels);
    this.problem_train_skip = createProblem(train_skip,
        train_labels);
    this.problem_tune_skip = createProblem(tune_skip, tune_labels);
    this.problem_train_abstr = createProblem(train_abstr,
        train_labels);
    this.problem_tune_abstr = createProblem(tune_abstr, tune_labels
        );
    this.problem_uni = createProblem(train_uni, tune_uni, labels);
    this.problem_skip = createProblem(train_skip, tune_skip, labels
        );
    this.problem_abstr = createProblem(train_abstr, tune_abstr,
        labels);
    this.train_abstr = train_abstr;
    this.tune_abstr = tune_abstr;
}

/**
 * Train all models
 */
public void train() {
    uniModel = solveProblem(problem_uni, problem_train_uni,
        problem_tune_uni, 0, 1000, 1000);
    skipModel = solveProblem(problem_skip, problem_train_skip,
        problem_tune_skip, 1, 20000, 20000);
    abstractModel = solveProblem(problem_abstr, problem_train_abstr
        , problem_tune_abstr, 2, 2, 2);
}

/**
 * Print results of best models
 * @param fold number of fold
 */
public void showResults(int fold) {
    System.out.println(Main.GREEN + "> Results of fold " + fold +
        Main.RESET);
    String[] setups = {"Unigram (linear)", "Skip-bigram (linear)",
        "Final (linear)", "Unigram (rbf)", "Skip-bigram (rbf)", "
```

```
        Final (rbf"};
    for (int i = 0; i < 3; i++) {
        System.out.println(Main.GREEN + "\t" + setups[i] + Main.
            RESET + "\n\tF1 = " + f1s[i] + "\n\tC = " + cs[i]);
        System.out.println(Main.GREEN + "\t" + setups[i+3] + Main.
            RESET + "\n\tF1 = " + f1s[i+3] + "\n\tC = " + cs[i+3] + "
            \n\tgamma = " + gammas[i]);
    }
}

/**
 * Solve an svm problem by training a linear and an RBF model and
    return the model with the highest F1
 * @param problem problem to solve
 * @param train_problem train problem
 * @param tune_problem tune problem
 * @param run 0 = unigram; 1 = skipBigram; 2 = abstract model
 * @param delta_gamma max difference in gamma value
 * @param g_step stepsize to increment gamma value
 * @return a trained model
 */
private svm_model solveProblem(svm_problem problem, svm_problem
    train_problem, svm_problem tune_problem, int run, int
    delta_gamma, int g_step) {
    int n_features = train_problem.x[0].length;
    // Train with linear kernel
    svm_model linearModel;
    int best_c = 15;
    Double f1 = 0.0;
    double gamma = 1 / (double) n_features;
    for (int c = 10; c > 1; c--) {
        linearModel = svm.svm_train(train_problem, svm_parameters(
            svm_parameter.LINEAR, c, gamma));
        ArrayList<Double> predictions = new ArrayList<>();
        for (int i = 0; i < train_problem.x.length; i++)
            predictions.add(svm.svm_predict(linearModel,
                train_problem.x[i]));
        Double f1_new = computeF(predictions, train_labels);
        if (f1_new.compareTo(f1) > 0) {
            f1 = f1_new;
            best_c = c;
        }
    }
    f1s[run] = f1;
    cs[run] = best_c;
    linearModel = svm.svm_train(train_problem, svm_parameters(
        svm_parameter.LINEAR, best_c, gamma));

    // Train with RBF kernel
    svm_model rbfModel;
    best_c = 15;
    f1 = 0.0;
```

```
    gamma = 1 / (double)(n_features-delta_gamma);
    double best_gamma = gamma;
    for (int g = -1 * delta_gamma; g <= delta_gamma; g += g_step) {
        gamma = 1/(double)(n_features+g);
        for (int c = 7; c > 0; c--) {
            rbfModel = svm.svm_train(train_problem, svm_parameters(
                svm_parameter.RBF, c, gamma));
            ArrayList<Double> predictions = new ArrayList<>();
            for (int i = 0; i < tune_problem.x.length; i++)
                predictions.add(svm.svm_predict(rbfModel,
                    tune_problem.x[i]));
            Double f1_new = computeF(predictions, tune_labels);
            if (f1_new.compareTo(f1) > 0) {
                f1 = f1_new;
                best_c = c;
                best_gamma = gamma;
            }
        }
    }
    f1s[run+3] = f1;
    cs[run+3] = best_c;
    gammas[run] = best_gamma;
    rbfModel = svm.svm_train(problem, svm_parameters(svm_parameter.
        RBF, best_c, best_gamma));

    // Return best model but save both
    if (f1s[run] >= f1s[run+3]){
        try {
            svm.svm_save_model("model" + run + ".txt", linearModel);
            svm.svm_save_model("model" + run + "_alt.txt", rbfModel)
                ;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return linearModel;
    } else {
        try {
            svm.svm_save_model("model" + run + "_alt.txt",
                linearModel);
            svm.svm_save_model("model" + run + ".txt", rbfModel);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return rbfModel;
    }
}

/**
 * Create a set of parameters
 * @param kernel the kernel to be used
 * @param c c parameter
 * @param gamma gamma parameter
```

```
 * @return svm parameters
 */
private final svm_parameter svm_parameters(int kernel, int c,
    double gamma) {
    svm_parameter param = new svm_parameter();
    int[] weight_labels = {0, 1};
    double[] weights = {1,1};
    param.weight_label = weight_labels;
    param.weight = weights;
    param.cache_size = 100;
    param.eps = 0.001;
    param.nr_weight = 2;
    param.nu = 0.5;
    param.p = 10;
    param.shrinking = 0;
    param.probability = 1;
    param.svm_type = C_SVC;

    param.kernel_type = kernel;
    param.C = c;
    param.gamma = gamma;

    return param;
}

/**
 * Compute the F1 score
 * @param predictions list of predictions
 * @param labels list of labels
 * @return F1 score
 */
private double computeF(ArrayList<Double> predictions, double[]
    labels) {
    int tp = 0, fp = 0, fn = 0;
    for (int i = 0; i < labels.length; i++) {
        Double label = labels[i];
        Double pred = predictions.get(i);
        System.out.println(Main.RED + pred + " - " + label + Main.
            RESET);
        if (label.equals(pred)) {
            if (label.compareTo(0.5) > 0)
                tp++;
        }
        else {
            if (label.compareTo(0.5) > 0)
                fn++;
            else
                fp++;
        }
    }
    double precision = tp / (double)(tp + fp);
    double recall = tp / (double)(tp + fn);
```

```
    double f1 = 2 * ((precision * recall) / (precision + recall));
    return Double.isNaN(f1) ? 0.0 : f1;
}

/**
 * Apply a model to a new dataset
 * @param users list of user in test set
 * @param uni list of unigram frequencies
 * @param skip list of skip-bigram frequencies
 * @param abstr list of abstract features
 * @return list of confidence values
 */
public ArrayList<Double> applyModel(ArrayList<ArrayList<String>>
    users, ArrayList<ArrayList<Double>> uni, ArrayList<ArrayList<
    Double>> skip, ArrayList<ArrayList<Double>> abstr) {
    double[] true_labels = new double[users.size()];
    for (int i = 0; i < true_labels.length; i++)
        true_labels[i] = Double.parseDouble(users.get(i).get(2));
    svm_problem uniProblem = createProblem(uni, true_labels);
    svm_problem skipProblem = createProblem(skip, true_labels);
    svm_problem abstrProblem = createProblem(abstr, true_labels);

    ArrayList<Double> predictions_uni = new ArrayList<>();
    ArrayList<Double> predictions_skip = new ArrayList<>();
    for (int i = 0; i < uniProblem.x.length; i++) {
        double uni_pred = svm.svm_predict(uniModel, uniProblem.x[i])
            ;
        predictions_uni.add(uni_pred);
        double skip_pred = svm.svm_predict(skipModel, skipProblem.x[
            i]);
        predictions_skip.add(skip_pred);
    }

    for (int i = 0; i < abstrProblem.l; i++) {
        abstr.get(i).add(predictions_uni.get(i));
        abstr.get(i).add(predictions_skip.get(i));
    }

    ArrayList<Double> predictions_abstr = new ArrayList<>();
    for (int i = 0; i < abstrProblem.x.length; i++) {
        double[] probs = new double[abstractModel.nr_class];
        double prediction = util.svm.svm_predict_probability(
            abstractModel, abstrProblem.x[i], probs);
        System.out.println(Main.RED + probs[0] + " - " + probs[1] +
            Main.RESET);
        predictions_abstr.add(prediction);
    }

    return predictions_abstr;

}
```

```
    @Override
    public void run() {
        train();
    }

}
```