Leopold-Franzens-University Innsbruck

Institute of Computer Science
Databases and Information Systems

Development of a Hack Detection System for Twitter based on Crowd Reaction Analysis

Master Thesis

Benjamin Murauer BSc.

supervised by
Dr. Eva Zangerle
Prof. Dr. Günther Specht

Innsbruck, May 11, 2016

Leopold-Franzens-Universität Innsbruck

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eingereicht.

Datum	Unterschrift

Abstract

Twitter, a popular microblogging platform, is often targeted by hackers who take over accounts in order to send spam. This triggers a change not only in the affected accounts behavior itself, but also often in the network of users connected to it. This thesis analyzes the possibilities of gathering information about hacked Twitter accounts by searching for messages that contain suggestions of an account being hacked. We then analyze the response of the alleged hack victims to determine how often such users respond to hints from their environment. Additionally, a qualitative analysis shows the different types of conversations that have been discovered this way.

Zusammenfassung

Twitter ist eine beliebte Microblogging-Plattform und wird bedingt durch seinen Erfolg auch oft das Ziel von Hackern, die über manipulierte Accounts Spam verschicken. Wenn Benutzeraccounts gehackt werden, löst dies nicht nur Änderungen im Schreibeverhalten der betroffenen Accounts aus, sondern wirkt sich auch oft auf das Verhalten der sozialen Kontakte der Betroffenen aus. Diese Arbeit untersucht die Möglichkeit, gehackte Twitter-Benutzer zu erkennen, indem das Verhalten des Umfeldes analysiert wird. Durch das Auslesen und Analysieren der Antworten auf etwaige Hinweise stellen wir fest, wie viele der betroffenen Benutzer auf Hinweise aus ihrem Umfeld reagieren. Eine zusätzliche qualitative Analyse erläutert eine Vielzahl an verschiedenen Hinweisen und deren Reaktionen.

Contents

Chapter 1

Introduction

With social media usage rising constantly, more and more user accounts are being hacked. The main goal of hacking social media is to spread malicious content, mostly by sending links to spam sites. Taking control over existing, valid users has many advantages for a hacker in comparison to creating dedicated spam accounts, since they already have a trustworthy connection to their peers [KKL$^+$08].

Twitter, a microblogging service, is no exception to this trend, as many of its users have to face account theft or abuse. Although Twitter itself has some very effective measures against spamming, especially for detecting dedicated spam accounts, still many malicious messages are being sent. Especially when users are not very active, they might need some time until they detect the misuse of their user accounts. Though many previous approaches already detect hacked accounts very successfully by analyzing the content of the alleged spam, little research is done on the peers of the hacked accounts.

We analyze the communication between allegedly hacked user accounts and their relationships on Twitter and find that many users that detect suspicious activities from their contacts actually ask whether their account was hacked or suggest it to have been so. By fetching the response of these alleged victims, we analyze how many of those mentioned accounts react in any way to these hints.

Our approach starts with a set of messages gathered by the DBIS research group of the University of Innsbruck. In a first step, messages are selected that suspect other Twitter accounts of being hacked. This is achieved by using a TF-IDF vectorizer for feature extraction and a support vector machine as a classifier. Additionally, some linguistic features such as stemming, lemmatization and stop word removal are tested for suitability. We then try to reconstruct the reaction of the possibly affected user and analyze it both quantitatively and qualitatively. To do so, we fetch possible responses of the affected accounts using the Twitter

APIs. The responses are then classified using the same methods as in the first step.
It shows that 30% of the users that have been asked whether their account was hacked responded positively, either confirming the suspicion or explaining the situation. We also perform a qualitative analysis on a selected set of messages, reconstruct the communication path, explain different scenarios and consequently give an overview of some different possible cases that are covered by our approach.
The remainder of this thesis is structured as follows: Chapter 2 explains the environment of Twitter and which parts of it are of importance for this thesis. It also contains background information on machine learning techniques and linguistic aspects that are used. Related work is listed in Chapter 3. Afterwards, a detailed scheme of the methods used is shown in Chapter 4 and its implementation is explained in Chapter 5. A qualitative analysis is given in Chapter 6, where an overview of the different types of conversations is explained. The results are displayed in Chapter 7 along with a discussion and references to possible future work.

Chapter 2

Background

Analyzing the behaviour of social media users often depends on features of the specific media used. Since the topic of this thesis is Twitter, the basic functionalities and characteristics of this social media service are explained in this chapter.
The second part of this chapter focuses on the machine learning methods that are used and explains the concepts behind them.

2.1 Twitter

Twitter is a popular microblogging platform that was launched in July 2006. With an active user base of 316 million users[1], it is the largest microblogging service. Its fundamental service is sending short messages, called *tweets*, that have a maximum length of 140 characters. In the remainder of this thesis, the terms message and tweet are used synonymously. As of March 2015, over 500 million messages are sent via Twitter each day [abo].
Generally, each message sent via Twitter is public and could be read by anyone. Exceptions to this rule are private profiles, who only share their tweets to their followers.
To add more context to the 140 characters, users have various possibilities. Some of those are triggered by entering specific strings into the message itself. Twitter parses these strings and internally reacts to their meanings.

2.1.1 Mentions

Mentions are entered by writing an @-symbol followed by the screen name of a user on Twitter. This way, the message will appear on the front page of the mentioned user, regardless of whether the two users are

[1]`https://about.twitter.com/company`

connected with a follower-followee relationship (which will be explained shortly).

2.1.2 Hashtags

Hashtags are used to categorize tweets and are used by users to contribute content to a specific topic. They can consist of any word that has the #-symbol prepended, without a space separating them. This way, users can search for any hashtag and find information related to it. It is possible to include more than one hashtag in a tweet.

> *Wayne Rooney has scored 30 goals in the #ChampionsLeague, more than any other English player.*

Example of a tweet regarding the Champions League

2.1.3 Retweets

If a user wants to spread the information of an existing message, one can forward the tweet. As [BGL10] denotes, there are multiple methods of classifying a message as a retweet, the most common one is prepending the message with the sequence `RT` followed by a mention to the original poster. Often, the message retweeted must be changed by the retweeting user due to the 140 character limit.

> *RT @montitan: Great game by @FNATIC! #LCS*

Example of a retweet containing a mention and a hashtag

2.1.4 URLs

To be able to post links in tweets, Twitter automatically uses its own URL shortening service to reduce the length of each URL posted. Initially, an external shortening service was used (first TinyURL, then bit.ly [twib]) before their own implementation was introduced in 2011 [tco]. It is not possible to prevent the URL from being shortened by Twitter. The original destination is however stored and contained in the tweets that are available via the API. It reduces the length of any URL to exactly 23 characters, even when the original URL is shorter [twic].

> *https://t.co/smw1g1gqGq*

Example of URL that has been reduced by Twitter

Besides adding context to the tweet itself, Twitter also offers methods to direct the information flow more directly. These are not part of the tweet itself.

2.1.5 Direct Messages

Tweets are not suited for sending sensible information, as they are generally public. Users can send private messages to other users, which are called *direct messages* on Twitter and can not be fetched by the public APIs. As of August 2015, Twitter removed the previously existing limitation of 140 characters for direct messages [twid]. They are often used for spam, as people generally trust direct messages more than public tweets, according to [KKL$^+$08].

2.1.6 Following Users

A user can follow another user to be kept updated with this user's tweets. This relation is unidirectional, but can be set up for both users separately. Neither Twitter nor the English language feature an antonym for the word *follower*, but in other papers the word *followee* has established itself and will also be used in the remainder of this thesis. When a user opens the front page on Twitter, a list of messages of the user's followees is displayed. Users can decide to generally mark their tweets as private, which causes them to only be visible for the followers of the author. Otherwise, all messages are public and can be found using the search or any kind of API.
User accounts with many followers are therefore more attractive for hackers, as they offer a larger audience for alleged spam. This way, hackers have a higher chance of distributing malicious content, as acquiring many followers using dedicated spam accounts is more difficult [TGSP11]. This makes celebrities and companies with many followers attractive targets for hackers.

2.1.7 Authentication

To prevent abusive behavior, Twitter requires all applications using the APIs to be authenticated. Two different authentication methods are offered, which both rely on the OAuth authentication framework[2]. Table 2.1 shows different application functionalities and their required authentication level.

User based authentication

User based authentication provides the application access to all user-related functions, including posting tweets and searching for users. It is generally meant for applications that are used by a single user, like alternative clients. Connecting to the Stream API requires user based authentication. Note that the access to direct messages that is noted in

[2]`http://oauth.net`

application authentication	user authentication
• fetch user timelines • access friends and followers from any user account • retrieve information of a specific user • search in tweets	• post tweets • connect to streaming endpoints • search for users • access DMs

Table 2.1: Authentication Methods and Some of Their Available Functions

the table does only provide insight in the messages of the authenticated user, not others.

Application based authentication

If an application does not need any of the functions that require user authentication, it can authenticate on behalf of the application itself. The limited access to functions is made up with higher rate limits for the REST API. Despite the naming, also the application based authentication method requires the application author to have a Twitter Account.

2.1.8 APIs

Twitter allows third parties to collect the public messages using two different APIs. Each API requires the application using it to authenticate itself.

Streaming API

Twitter's Streaming APIs allow programs to collect messages in a continuous way. A HTTP connection is kept alive for a stream and messages are transmitted with low latency. All streaming APIs feature various filter mechanisms like searching for specific words within the message content or filtering messages by specific users. Twitter offers various pre-defined stream endpoints for specific use cases:

- The *Public Stream* returns messages from the entirety of Twitter's communication. For common applications, the fraction of messages returned is limited to 1% [Hue]. If an application requires more data, one has to purchase a higher access level from Twitter. Third parties like GNIP[3] and DataSift[4] used to sell customized data rates until Twitter bought GNIP in 2014 [Mes] and terminated all relationships to other enterprise resellers [Bry].

- The *User Stream* returns messages by a single user and roughly represents the activity on Twitter by that user.
- A *Site Stream* can be seen as a combination of many user streams, suited for applications that need to examine multiple users at once. As of February 2016, this feature is not yet fully deployed and still in a development stage.

REST API

The REST API is able to answer to specific requests, making it more flexible but also slower than the streaming API access. It can be used to fetch tweets sent in the past or to check if a certain user still exists. Applications are restricted in how many requests they can send to Twitter per 15 minute time slot.

2.2 Machine Learning

To be able to classify a large amount of messages automatically, the concept of machine learning is used. Generally, two different types of machine learning tasks can be distinguished.
Unsupervised learning can be used to extract information from a set of samples without using previously obtained knowledge [Bis06]. Predicting a category for unlabeled information is called clustering.
Supervised learning methods use previously gathered information about the data to predict this information on untrained data. Category prediction on labeled data is called classification, which is performed in this thesis. Figure 2.1 depicts various machine learning techniques offered by the `scikit-learn`[5] python library and their preferred application area.

2.2.1 Feature Selection

A feature can be seen as a measurable property of a sample. Choosing the correct features heavily depends on the problem at hand and often requires in-depth knowledge about the problem's domain. When

[3]`https://www.gnip.com/`
[4]`https://www.datasift.com`
[5]`http://scikit-learn.org`

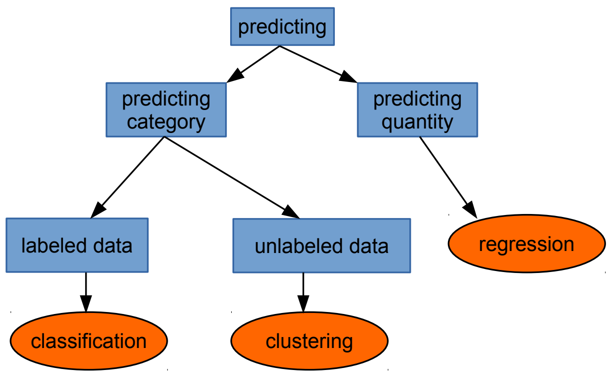


Figure 2.1: Choosing the Correct Machine Learning Technique, Simplified From scikit-learn

	"hello world"	"Bob says hello"
hello	1	1
world	1	0
Bob	0	1
says	0	1

Table 2.2: Simple Features Extracted From Text Messages

comparing samples, a machine learning algorithm usually uses multiple features of each sample that are combined in a feature vector. The program part that calculates these vectors is called a *vectorizer*. When classifying text messages, the content of the text obviously is of great importance. A simple model for a feature vector would be to count the occurrence of words contained in the text, as shown in Table 2.2. This model, which is commonly referred to as "bag of words" [Bis06], ignores the word order, which is an advantage for the scenario of this thesis, as the English grammar allows multiple ways of writing a sentence: "was your account hacked?" and "did somebody hack your account" have a different word order, but both suggest a similar scenario. A bag of words feature vector would be able to show this similarity, as the vector for both sentences contain values for the words "your", "account" and "hacked". Of course, this is a very rough model. It cannot detect negation and is susceptible for denormalization of the vectors; words that occur in many sentences do not necessarily add any information. For example, every message in the dataset used contains the word "hacked" or "account", as these were the keywords used by the Twitter crawler.

TF-IDF

The TF-IDF model [SWY75], which stands for *term frequency / inverse document frequency*, takes these facts into account. It emphasises the importance of words that occur often in a single sample, but lowers it if it occurs often in the complete document set. There are multiple ways of computing TF-IDF that mainly differ in the normalization methods used. The most basic (non-binary) variant can be described as:

$$tfidf(t,d,D) = tf(t,d) \cdot idf(d,D) = tf(t,d) \cdot log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

where t denotes the term calculated, d the document (e.g. message) and D the entire document set. $tf(t,d)$ is the number of times t occurs in d, $|D|$ describes the size of the entire corpus and $|\{d \in D : t \in d\}|$ describes the number of documents that contain t. Note that division by zero is not possible, as a term is always contained in at least one document. The implementation used by `scikit-learn` uses a slightly modified version of this formula:

$$tfidf(t,d,D) = tf(t,d) \cdot \left(1 + log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)\right)$$

This way, even when a term occurs in every single document, its TF-IDF score is still non-zero. A "term" doesn't have to be a single word but can be an arbitrary string, possibly consisting of multiple words. This is one of the configurable parameters called n-grams that affect the behaviour of the TF-IDF vectorizer. The complete set of parameters and the values that were used are explained in Chapter 5.

Other Features

Feature selection is a domain-specific problem, images for example have intuitively different features than text documents. In case of Twitter text analysis, one could build a vectorizer that adds information about hashtags, mentions and URLs. Table 2.3 shows an example of Twitter-specific features along with the bag-of-words features. Different feature vectors can be combined by concatenating their values and can be emphasised by applying different weighting schemes to them. However, this approach exceeds the scope of this thesis.

2.2.2 Vector Distances

To be able to measure the similarity of two vectors, a suitable measure has to be used. In literature, a set of commonly used measures are L^p-norms, which are defined as

$$L^p(x) = ||x||_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{x}{p}}$$

Text A:
@alice have you been hacked? #badpw

Text B:
http://t.co/foobar @bob yes i was hacked

	Text A	Text B
have	1	1
you	1	0
been	0	1
hacked	1	1
yes	0	1
i	0	1
was	0	1
links	**0**	**1**
mentions	**1**	**1**
hashtags	**1**	**0**

Table 2.3: Twitter-Specific Feature Extraction

The most commonly used are the *Manhattan* norm ($p = 1$) and the *Euclidean* norm ($p = 2$). For simplicity, these two norms are often referred to as $l1$ and $l2$ norms.
As of the features themselves, vector distance norms may be defined depending on the problem domain. For example, a Twitter-specific distance could be to calculate the $l2$ norm for the TF-IDF features and a $l1$ norm for the Twitter-specific features and then combine them. Choosing the best norm is an optimization task that is not trivial to answer beforehand and often is searched for automatically, among with other parameters. However, this thesis does not use self-implemented vector distances but relies on the built-in $l1$ and $l2$ distances offered by `scikit-learn`.

2.2.3 Feature Filtering

When dealing with text classification, the feature vectors can get very large, as every possible word is a potential feature. To decrease the size of these vectors and therefore the computational costs of handling these vectors, features are often filtered. This means that only the most valuable features are kept in the vector while the non-relevant ones are removed. Determining which features are relevant can be done in many ways (e.g. document frequency, information gain, χ^2-test, term strength or others). [YP97] Shows that a χ^2-test can be very effective in reducing text features. Reducing the number of features is often desired for performance reasons, but can also decrease the performance of the classifier, since it essentially reduces the amount of information available. Therefore, filtering features is difficult to recommend in general but rather

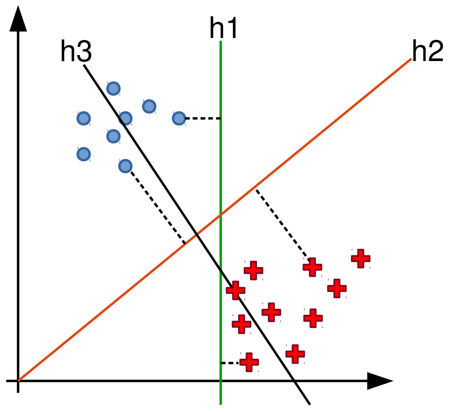


Figure 2.2: Example of Hyperplanes and a Set of Samples

depends on the problem at hand.
This thesis uses a χ^2-test, which is implemented in the `scikit-learn` library. For each feature-class combination, it calculates the probability of a feature being independent of the class. The lower this probability is, the more important the feature is for the model. After assigning a probability for each feature, the top-n features are selected for further processing. Choosing n is an optimization task that can be handled by e.g. a brute force grid search approach.

2.2.4 Support Vector Machines

Once the feature vectors are extracted from all documents, they can be used by a *classifier* for future document prediction. Many different classifier families exist, all with their own set of strengths and weaknesses, which have to be considered when choosing a classifier. For example, some classifiers don't perform great with high dimensional feature vectors.
A Support Vector Machine or SVM is a model introduced by Boser, Guyon and Vapnik in 1992 [BGV92], but the underlying research had already been going since the 1930s [Fis36]. Research [Joa98, DPHS98] has shown that SVMs are suited for challenges that text classification brings, mainly the high dimensionality of the feature vectors.
Roughly speaking, a SVM tries to construct a hyperplane that separates samples, while trying to maximize the overall border distance to them. An example for such planes can be seen in Figure 2.2, which shows an example of a SVM in a two dimensional space.
While $h3$ does not separate the blue (circled) and red (crossed) samples at all, $h2$ does it with a bigger margin than $h1$. The samples that define the margin are called *support vectors*.

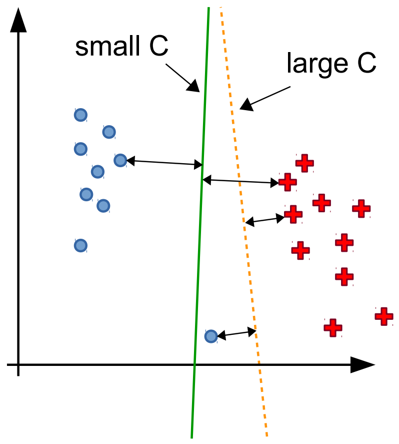


Figure 2.3: Different Values for C

The technical definitions of support vector machines are out of the scope of this thesis and can be read in [Vap95]. One feature that is relevant for this paper is the *strictness* parameter C. The higher the value of C, the more penalty the model puts on misclassifying a training sample. Figure 2.3 shows a continuously drawn hyperplane using a small C value, which misclassified one blue sample while training, but features a large margin. The dotted hyperplane that uses a large C does not misclassify the sample, but has a thinner margin. In general, it is not possible to say which scenario is better, since this depends on the data that is to be predicted. Figure 2.4 shows two scenarios for different samples that have been predicted. It is clear that on the left side, the hyperplane with the smaller C performs better, while on the right side the larger C is the better choice. At training time, this cannot be foreseen.

Kernel Functions

Often, data is not linearly separable in the original input feature space, one example can be seen in Figure 2.5. One method that can be used by SVMs is to calculate implicit features of two vectors in a higher-dimensional feature space, where separation is easier. However, linear SVM implementations are much faster (e.g. `liblinear`) and text classification often does not necessarily benefit from higher dimension kernels [HCL03]. For these reasons, this thesis uses a linear kernel.

2.2.5 Unbalanced Classification

The optimal case for training a SVM requires the data to be distributed evenly among the available classes. Naturally, this is not the case for

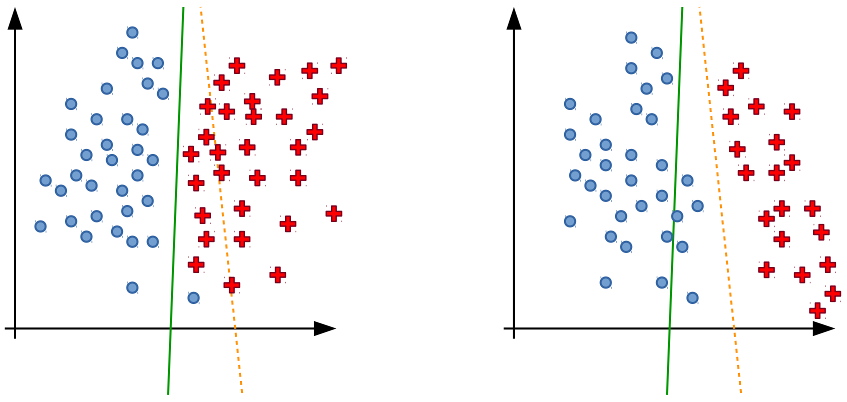


Figure 2.4: Different Scenarios for Prediction Data

many real life situations. When a class is underrepresented, as is displayed on the left side of Figure 2.5, the hyperplane will generally be too close to the smaller class. There are various ways that cope with this imbalance [Hos05], which can be separated into two general groups:

- Change the training samples to be evenly distributed. This can be achieved by different techniques:
  - Gather more data for the underrepresented class. Mostly, this is the preferred method, as it also adds information to the classifier.
  - Delete samples from the overrepresented class (undersampling). This method can only be used if the remaining samples still represent the class well enough [KM97].
  - Generate dummy samples for the underrepresented class (oversampling). This can be done by plainly copying samples or by generating artificial samples by introducing noise.

  In most cases, gathering more data is the preferred option as it increases the total amount of information available to the model. Research whether over- or undersampling yields better results in general are inconclusive [Hos05]. It generally depends on the problem which approach performs better.

- Modify the classification process by tuning the SVM. In this case, the SVM can use different values for C for the two classes, putting a higher penalty on misclassifying the underrepresented one. This option is built-in in the `scikit-lern` library, which normalizes the C value according to the number of samples in each class. A schematic outcome of this method is displayed by the dashed line of Figure 2.5.

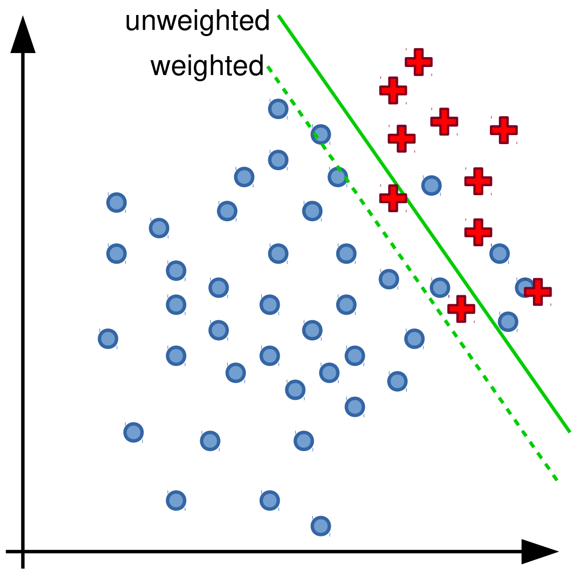


Figure 2.5: Unbalanced Classification with and without Class Weights

2.2.6 Cross Validation

A commonly used technique to verify parameters in machine learning is cross validation [AC+10,sci]. Multiple methods exist, sharing a common basic strategy: a subset of samples is excluded from the set and is used for validation. The remaining samples are used for fitting the model (in case of a SVM: finding the hyperplane). Afterwards, the model predicts the classes of the previously separated verification samples. Then, these classes are compared to the true classes, which are known in advance, since they are manually classified. This step is then repeated using a different verification and training set.
Depending on whether all possible combinations of verification and training samples are actually tested, cross validation techniques can be distinguished into exhaustive and non-exhaustive methods. The most important methods include [AC+10]:

Exhaustive Cross Validation Methods

- Leave-p-Out: P samples are used for verification, the rest is used for training. This method is not suitable for a high number of samples, as the amount of possible combinations which have to be tested grows very quickly with C_p^n, where n is the number of total

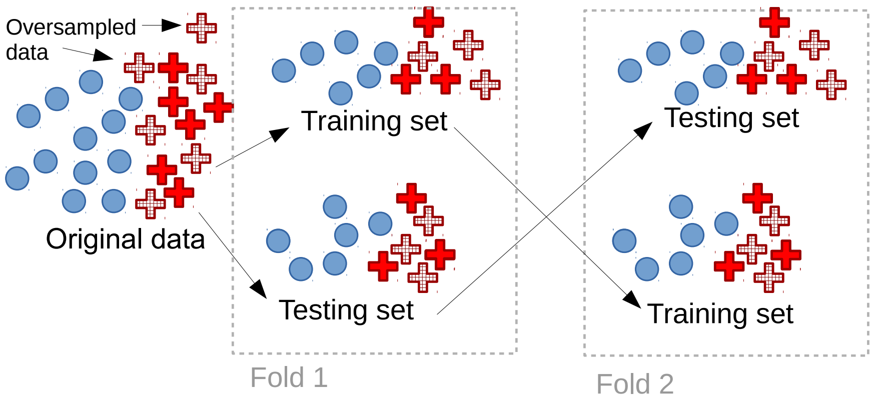


Figure 2.6: Example for Wrong Cross Validation With Oversampled Data

training samples.

- Leave-One-Out (LOO): This is a special case of leave-p-out with $p = 1$. Since $C_1^n = n$, this method is better suited for larger sample sets.

Non-Exhaustive Cross Validation Methods

- k-Fold: The total training set is split into k subsets. Each of the subsets is used for verification once, while the remaining ones are used for training.
- Stratified k-Fold: A special variant of k-Fold where the ratio of classes is tried be kept constant within the subsets.
- Repeated Random Sub-Sampling (Monte Carlo Cross Validation): This method selects training and verification sets randomly. Due to the randomness, some combinations may occur multiple times while others may never be tested. Depending on how many iterations of this method are used, its results approximate the leave-p-out method.

When using cross validation together with sampling, it is important that the folds that are used for testing do not contain sampled data. Otherwise, the respective score is calculated using data that does not represent the actual data at hand and will not give an accurate image of the classifier's performance for later prediction. An example for wrong and correct cross validation (with k=2) using oversampled data is shown in Figures 2.6 and 2.7. In a real life scenario, k is usually higher (i.e. 5 or 10).

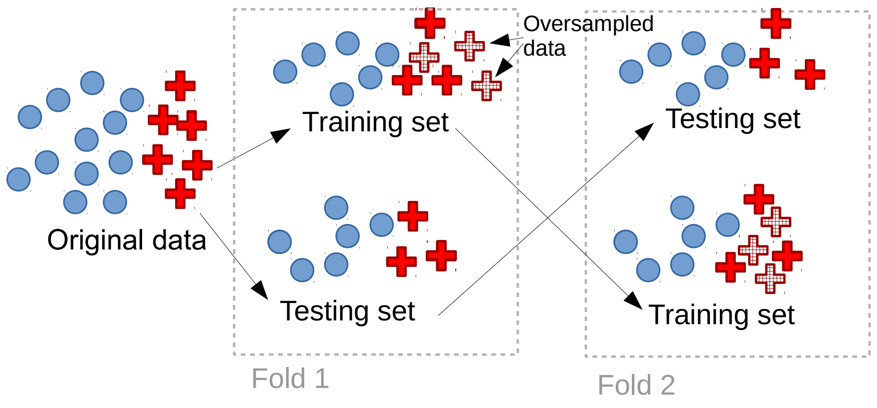


Figure 2.7: Example for Correct Cross Validation With Oversampled Data

		True class	
		True	False
Predicted class	True	True Positive	False Positive
	False	False Negative	True Negative

Table 2.4: Confusion Matrix

2.2.7 Quality Measures

There are several different methods to measure the quality of predicted results. For the following explanations, TP stands for *true positives*, thus representing elements that were labelled correctly by the classifier. TN stands for *true negatives*, FP for *false positives* and FN for *false negatives*. The respected classes are explained by the confusion matrix in Table 2.4.

- *Accuracy* is defined as $\frac{TP+TN}{TP+TN+FP+FN}$ and gives an overview of how well a classifier is performing by analyzing the total of correct vs uncorrect predictions. It is an unsuitable score for unbalanced sets; Imagine a training set consisting of 1 positive and 10 negative samples. A prediction model that blindly classifies every sample as negative still reaches an accuracy score of $\frac{9}{10}$, but is obviously not suited for a real life application.
- The *precision* value is defined as $\frac{TP}{TP+FP}$, thus representing the fraction of the selected results being true. A perfect score of 1 means that no false positives have been selected. A synonym for precision is *positive predictive value*.
- The *recall* value is defined as $\frac{TP}{TP+FN}$, a high recall value therefore means that nearly all of the relevant samples have been matched

correctly by the classifier. It is also known as *true positive rate*, *hit rate* or *sensitivity.* A score of 1 means that all positive samples have been predicted as such.

- The *F1-score* is defined as $\frac{2TP}{2TP+FP+FN} = 2 \cdot \frac{precision \cdot recall}{precision+recall}$. It is the harmonic mean of precision and recall. As the F1-score does not take true negatives into account, this score is primarily used for *detection* tasks, which focus of the true detection of one class. It is a special case of the F_β score (with $\beta = 1$), which additional has a parameter β that determines which of the two sub-scores *precision* or *recall* has a higher weight.

Obviously, a trade-off between these scores has to be achieved, especially between precision and recall. Optimizing for precision usually means increasing the "strictness" of the classifier, which on the other hand, leads to an increase of false negatives, therefore lowering the recall value and vice versa. All of these measures have in common that their perfect score is 1, while a score of 0 is worst.
The relationship between precision and recall makes their relationship to each other a separate measure, called a *precision-recall* or *PR*-curve. In these curves, the relationship between precision and recall is displayed for different probability thresholds. Typically, a step-like pattern can be seen in PR-curves, which originates from the nature that precision and recall are defined.

- *Precision*: when lowering the classifier's threshold (for samples to be classified as positive), the total amount TP + FP (the denominator in precision) increases along with the TP (the enumerator). This means that lowering the threshold may increase or decrease precision: If the old threshold was too high, the new threshold causes more TPs to be detected than FPs, increasing precision. However, if the threshold was correct or too low already, the increasing FPs will cause precision to fall.

- *Recall*: When lowering the classifier's threshold, the amount of TPs rises, but the number of FNs decreases at the same rate. This means that recall may rise or stay constant when decreasing the threshold.

Together, these properties result in a step-like pattern: A small change in the threshold may cause the precision (on the y-axis) to drop rapidly, as many false positives are classified with this new threshold. Further changing the threshold results in TPs being classified as such, increasing precision. On the x-axis, the recall value does not fluctuate in such a way. Figure 2.8 shows an example of a PR-curve with this step-like appearance.

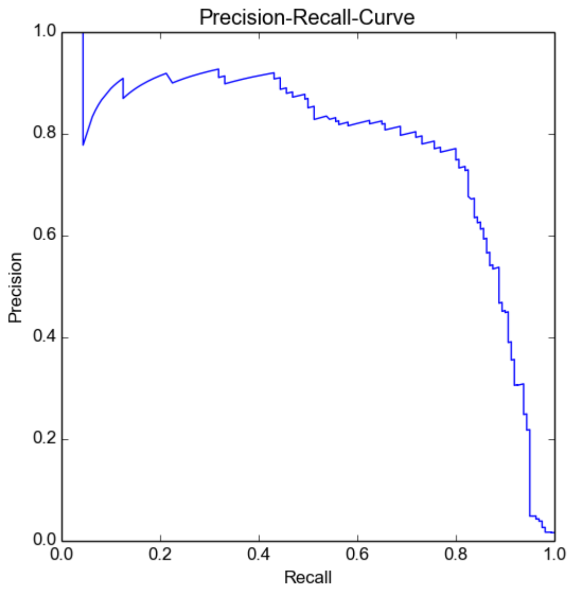


Figure 2.8: Example of a Precision vs Recall Curve

2.3 Linguistic Aspects

Handling text messages offers unique possibilities to improve classification. The tweets at hand all are written in the English language, as this was filtered at crawling time. This allows for specific improvements regarding the preparation of the samples. This section describes three linguistic improvements that have been tested while analysing the tweets.

2.3.1 Stop Words

Stop words are words that occur very often and don't add any semantic information to the text itself, typical examples are "a", "the" or "and". They can be removed to allow for smaller feature vectors and an improved performance of the analysis.

2.3.2 Stemming

Stemming aims to shorten words to a common word stem. This can be done by using a lookup table, by following simple rules that cut off or replace specific letters or by a combination of these techniques. The resulting stem does not necessarily have to be a valid word on its own. The words "argue" and "argued" could be stemmed to the common stem "argu".

Word	Stem
quickly	quickli
suspicious	suspici
hacked	hack

Table 2.5: Examples of Words Stemmed by the Porter Algorithm

Word	Stemmed	Lemmatized
better	better	good
ran	ran	run
been	been	be
was	wa	be

Table 2.6: Examples of Stemmed and Lemmatized Words

There are many different types of stemming implementations that differ in their set of rules. A commonly used implementation in computer linguistics is the *Porter Stemmer* [Por80]. Its author, Martin Porter, developed a complete stemming framework called *snowball*[6] including a custom stemmer definition language to easily write new stemmers for different languages. It uses a set of rules to shorten a word and classifies the length of words by introducing a custom interpretation of a syllable. Some examples for words stemmed by the Porter algorithm are listed in Table 2.5.

2.3.3 Lemmatization

Lemmatization is similar to stemming, but can handle more complex word relationships such as irregularity in verbs. The lemma of a word can be interpreted as its meaning [Leh] or how the word would be represented in a dictionary. While stemming can be seen as a cooking recipe to remove letters by following specific rules, lemmatization requires a larger knowledge of the analyzed language. Therefore, it is nearly impossible to construct a set of rules to apply to a word to lemmatize it. Using a dictionary provides more flexibility, which is required depending on the complexity of the language used. Table 2.6 shows a list of example words that have different stemmed and lemmatized forms. In contrast to stemming, lemmatizing always produces valid words as an output.

Lemmatization can be ambiguous, the word “meeting” can be interpreted as a verb or noun and lemmatizing it would lead to different words (“meeting” for noun, “meet” for verb). When trying to optimize

[6]`http://snowball.tartarus.org`

the results, such context has to be provided in addition to the word itself. However, supplying context often is not possible in an automated process. In this case, a word is often lemmatized in all possible variations (e.g. as verb and as noun) and the shortest outcome is used as the result.
Since stemming and lemmatizing both aim to find a common root of a word, it is intuitively not useful to apply both techniques to the same text, as the outcome depends on the order in which the operations have been performed. Stemming before lemmatizing could transform the word “quickly” to “quickli” to “quickli” (as “quickli” is not a valid word and cannot be lemmatized), but lemmatizing before stemming would lead from “quickly” to “quick” to “quick”.

Chapter 3

Related Work

The success of Twitter and its rising spam problems caused research to investigate many areas. Previous work includes many different systems for spam detection and classification. Spam messages can reach a Twitter user in different ways. [TGSP11] detect four major exposure locations:

User timeline. Spammers can post tweets on a users timeline by first luring them into a follower-followee-relationship or by using mentions, which does not require any previous interactions.

Trending topics. When using trending hashtags or keywords, the spam message is more likely to be in the result list of a query searching for those.

Search. Beyond trending topics, Twitter features a more specific search for message content. Spammers can include popular search terms for reaching a larger audience.

Direct messages. A user can receive spam by direct messages if the spamming account follows the user. The relationship does not need to be bidirectional for this communication channel to work.

3.1 Existing Spam Detection Methods

Because spammers are successful on Twitter, research on detecting compromised accounts has been going on for some time and many different approaches have evolved. The majority of the existing methods (e.g. [MC11, BMRA10, SKV10]) uses both message and user related information for classification:

Message Related Features

URLs. The main target of a spam message on Twitter is to redirect the user to illicit content, therefore many spam messages contain URLs. As described in Section 2.1.4, Twitter uses its own URL shortening service, making the final destination of a link invisible to the user. Spammers often use additional URL shorteners to obfuscate the true destination. A study performed by [TGSP11] shows that from over 1 billion distinct URLs collected, roughly 15 million distinct destinations were left after resolving one redirect (t.co was not yet launched). By using their own URL shortening service, Twitter can check the posted URL and detect blacklisted sites and malicious content before it is posted and react faster to spammers.
Retweets. As [GTPZ10] denotes, there are no limitations for retweeting messages on Twitter, making messages by popular accounts an easy target for hijacking:

> *http://spam.com RT @barackobama A great battle is ahead of us*
>
> Example hijacked Tweet, from [GTPZ10]

Hashtags. Some spammers use dedicated hashtags to lure users to alleged services:

> *Get more #fans, #followers, #customers http://bit.ly/1vWkQ5J*
>
> Example of dedicated spam hashtag

Others use trending topics in the spam message without relating to its meaning or content:

> *Help donate to #haiti relief: http://spam.com*
>
> Example of hijacking a trending topic, from [GTPZ10]

Temporal features. Most users have a temporally regular basis of posting, as [ESKV13] points out, including busy times (e.g. lunch break) or more quiet times (e.g. sleeping hours). A highly irregular posting pattern may indicate a malicious account.
Message content. The text itself can also be tested for containing suspicious terms, as spam mostly occurs around specific topics or in dedicated campaigns. The most popular topics according to [ESKV13] include free electronic media, jewelery, electronics, vehicles, finance or dieting.

Message language. Another indicator for spam is when a user is posting updates in a language that is different from the language spoken in the social relationships of that user.

Account Related Features

User relationships. Forming relationships on Twitter is difficult for Spammers. In the dataset observed by [TGSP11], 89% of the spam accounts have less than 10 followers, while 40% have none. Instead, these accounts use hashtags and mentions to distribute their content to as many users as possible.
Message rates. Sending messages with a high frequency is a typical indication for a spam account. In the research performed by [TGSP11], 34% of the observed spam accounts pursued the strategy of sending as many messages as possible before being suspended by Twitter. Other spamming accounts behave more carefully, sending messages only every few hours or days.

3.2 Twitter's Spam Detector

Twitter's methods for detecting and disabling compromised accounts are already quite effective. As [TGSP11] denotes, more than 77% of hacked accounts are being detected and suspended by Twitter within one day and 92% in three days. The details of the implementation of Twitter's detector are not public, but their rules [twig] give a good overview on what they are watching out for, including:

- a high frequency of following/unfollowing other users
- the messages consisting mainly of links
- a large number of other users blocking the observed account
- ...

3.3 Third Party Analysis

All the previously discussed methods have in common that they rely on data provided by the hacked account or message content itself. When working with this data, it has to be acquired as early as possible. When a user reclaims control over the hacked account, any unwanted messages sent during the compromised phase are often deleted, as is suggested by the Twitter guidelines [twia]. Once a message is deleted, it is impossible to retrieve using the public API. Therefore, one must collect tweets sent by the spammer to be able to use them for any data analysis.

People that follow the hacked account often might be able to notice before the original owner of the account does and post about the event, propagating the information. For the owner of the hacked account, this time difference may be of great importance.
This thesis investigates the possibilities of analyzing peer reactions to detect a possible account hack. This is done by watching accounts for suspicious user mentions that suggest an account being hacked. Afterwards, the messages that the mentioned users wrote after the mention are analyzed.
At the time of writing, this approach is unique in the research field.

Chapter 4

Methodology

Following previous work [ZS14], we use machine learning techniques to analyze public Twitter messages. The approach analyzed in this thesis consists of five basic steps that are visualized in Figure 4.1:

1. Preprocess and clean the crawled data (Sec. 4.1).
2. Find messages that contain suggestions of a Twitter account being hacked. This is achieved by using a Support Vector Machine that classifies tweets that were crawled by the DBIS institute (Sec. 4.2).
3. Filter those messages and ignore those which contain mentions of users that are no longer available on Twitter. This step involves using the Twitter API to search for user ids (Sec. 4.3).
4. Try to get possible responses from the mentioned users by using the Twitter REST API (Sec. 4.4).
5. Use a Support Vector Machine to identify possible responses as such (Sec. 4.5).

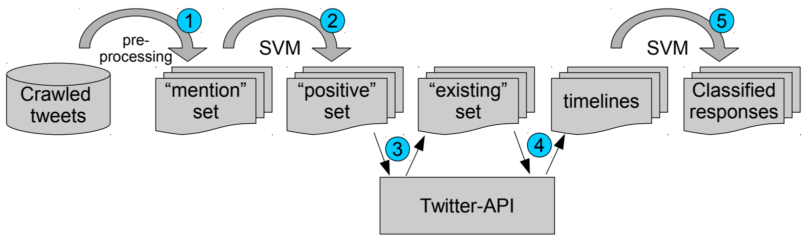


Figure 4.1: Basic Workflow

In this Chapter, the basic steps described above are explained. The implementation details and the usage of the scripts is covered in Chapter 5. Basic results of each main step are discussed in this chapter, as they explain the choices that were made for the following steps. The overall results are explained in Chapter 7.

4.1 Dataset

The basis of the analysis is a dataset collected by the DBIS research group of the University of Innsbruck [ZS14]. It consists of 4.7 million tweets that have been collected from November 2012 to October 2014 using the Twitter Filter API [twie]. All of these tweets contain either the words "hacked" or the word "account" in them. Statistical details of the dataset are shown in Table 4.1. Table 4.2 displays the most mentioned user names and the number of their occurrences. Celebrities like these can have a big impact on the dataset, as their messages are usually retweeted a lot. This thesis does not include any measures to detect or remove these duplicates.

4.1.1 Preprocessing

The first step was sorting out tweets that do not contain useful information for analyzing. This mainly includes two groups of messages: (1.) tweets that don't contain any mentions. As this thesis aims to collect third person data based on mentions, it is obligatory for each message to contain a mention, in particular as a json object field. All messages that do not contain such a field are removed from the set using a python script. (2.) tweets that don't contain the word "hacked". Many of the crawled tweets are retweets, see Table 4.1. Some Twitter clients offer a function to directly retweet a message and automatically crop the original message if the length of the new one exceeds the 140 character limit. By doing so, tweets may lose important information if the original message contained information about a possible hack:

> *RT @wayfaringcalum: @Calum5SOS hi cal,I hope youre having fun in America!! If you happen to see this pretty please refollow me,someone hack...*
>
> Example of a cropped tweet

These messages are fetched by the crawler because they store the retweeted message (which contains the word "hacked") in a separate json field. This message itself can not be used for further prediction, as the core content of the message is incomplete. However, the original tweet can

Attribute	Value
Tweets	4,698,845
Tweets incl. extracted retweets	5,984,406
Distinct authors	2,670,318
Retweets	1,495,325
Messages containing mentions	3,281,005
Messages containing hashtags	555,981
Distinct hashtags	120,690
Messages containing URLs	614,059
Distinct URLs	216,318

Table 4.1: Dataset Characteristics

be used.
There are other cases in which the information about the hack was added by the retweeting itself, often citing the alleged spam:

> *Who hacked ya account? "@blackboy_ken hurt so good - Carly Rae Jepsen"*
>
> Example of a retweet where the "hacked" information was added by the retweeter

In this case, only the retweet is valuable whereas the original message does not contain useful information for our purpose. To include all relevant messages, all available retweeted messages were extracted from the json information and added to the dataset. Afterwards, any message not containing the word "hacked" was removed from the dataset.
After performing these 2 filtering steps, the resulting set contains 2.2 million messages (see Table 4.3) and will be referred to as the *mention-set*.

4.2 Extracting the Positive-Set

Many messages were not of interest to the research question, as they either contained information about another type of account (e.g. email or banking account) or they were about the posting user and not about an other user (e.g. "my account got hacked"). The tweets that are of interest should suggest an other Twitter account being hacked (e.g. "@foobar has your account been hacked?"). A more detailed analysis of the data at hand can be found in Chapter 6. To be able to predict the relevance classes of the remaining tweets, machine learning techniques were applied.

User	Description	No. of Mentions
@NiallOfficial	singer	39,136
@donghae861015	singer	35,485
@Harry_Styles	singer	33,218
@justinbieber	singer	32,366
@Luke5SOS	singer	30,012
@twitter	official Twitter	21,285
@AllRiseSilver	singer	21,222
@Real_Liam_Payne	singer	17,445
@Michael5SOS	singer	16,153
@ArianaGrande	singer	16,130

Table 4.2: Most Occurring Mentions

Set	Messages
All	4,698,845
Mention	2,266,935
Positive	444,315
Existing	412,228

Table 4.3: Sizes of the Filtered Message Sets

The basic subworkflow for this step is displayed in Figure 4.2 and consists of following steps:

1. Manually classify a random subset of the document corpus.
2. Generate feature vectors by using a TF-IDF vectorizer.
3. Select the best features by filtering with a χ^2 test.
4. Train the SVM with the training vectors.
5. Predict the classes of the remaining documents.

The size of the resulting *positive*-set heavily depends on the parameters used for the vectorizer and classifier. To select these parameters, a brute-force grid search method was used, which is described in Chapter 5. When searching for optimal parameters, the grid search engine optimizes one measure, which must be specified. To select a suiting measure, the workflow as well as the problem at hand have to be analyzed:

- When analyzing the workflow, it becomes clear that the bottleneck of the steps at hand lies in the Twitter limitations in steps 3 and 4. Therefore, it would be computationally and temporally expensive

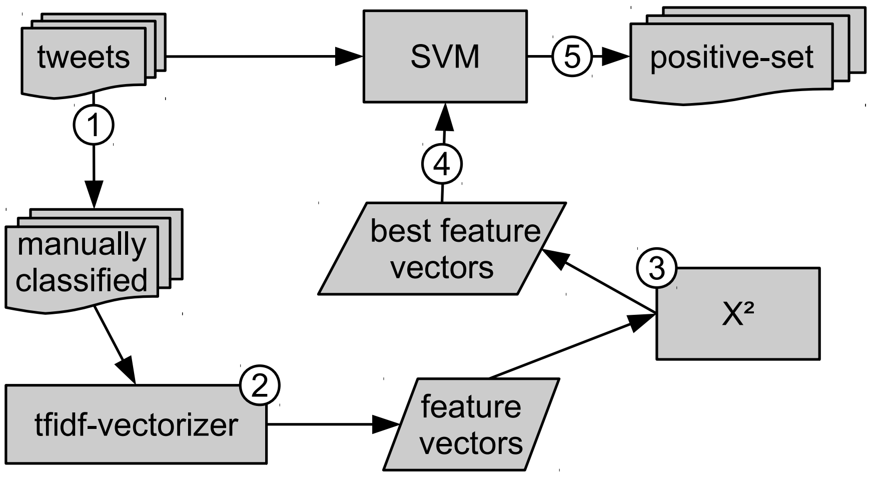


Figure 4.2: Class Prediction Workflow

to use valuable Twitter resources with false positive data, making the **precision** score the obvious choice for optimization. However, one can not only optimize for precision. Imagine a model that classifies one correct positive sample and predicts all others to be negative. This would yield a perfect precision score of 1.0, while being not usable at all.

- **Accuracy** is a suitable measure if the class sizes are equal, but even a small difference in training class size makes this approach less reliable. This phenomenon is called the *accuracy paradox*: a classifier that blindly classifies each sample to be of the majority class will achieve a score higher than 0.5 (which represents the score of a random classifier) without being of any use.
- The **f1**-score does not take true negative samples into account. However, when the classification aims to detect one specific class and disregards all other samples as "rest", this does not have any negative effect on the prediction. Therefore, the f1-score is the score that the grid search should optimize (for the *positive*-class, specifically).

Of the 3,650 messages that were classified manually, 455 were found positive. As described in Section 2.2.5, different methods exist for dealing with unbalanced data. Table 4.4 shows the f1-score of four methods that were tested for coping with the imbalance of the training samples. Figure 4.3 displays the recall vs precision curve of these methods.
It is clear that all four methods perform similarly, which can be explained with the rather small difference in class size. For its simplicity and the

method	description	f1-score
plain	does not consider the class imbalance at all	**0.73**
oversample	blindly copies random positive samples for the training set of the cross validation	0.70
undersample	randomly removes negative samples	0.72
autoweight	uses the built-in weighting mechanism of `scikit`	0.67

Table 4.4: Balanced vs. Unbalanced Training Class Sizes

slightly better f1-score, the *plain* version was used to calculate the final *positive*-set.

4.3 Filtering for Existing Users

Before the Twitter API is used to fetch the responses to these filtered messages, all tweets by users that are no longer active can be ignored. Twitter offers an own API call for looking up user names, which is more feasible than detecting these accounts by an empty response set. A batch of 100 user names can be looked up at once, with 180 requests per 15-minute time slot, resulting in looking up 10,800 users per hour. This way, the *positive*-set could be filtered in about 2 hours. The resulting message set, which can be seen in Table 4.3, shows that 9.2% of the users that were mentioned in the *positive*-set were no longer active or available.

4.4 Fetching User Timelines

Having determined the set of users still available, an obvious, interesting information is to know whether the corresponding user did reply to the mentioned tweet. For this purpose, the tweets posted right after the mention are of most interest.
Messages on Twitter have a dedicated field for if they are a direct reply to a previous message. An obvious method would be to use the search API to scan for direct replies to these mentions, but this would exclude general replies: often, users respond to all of their followers after reclaiming control over their account, rather than replying to a single reporter [ZS14]. As this is not a direct reply, the according field in the message would not be set and cannot be searched. Also, people often reply to a message without using the intended functionalities (e.g. when using a custom Twitter client), which results in the replies not showing up in the results from the appropriate API call. Therefore, all messages

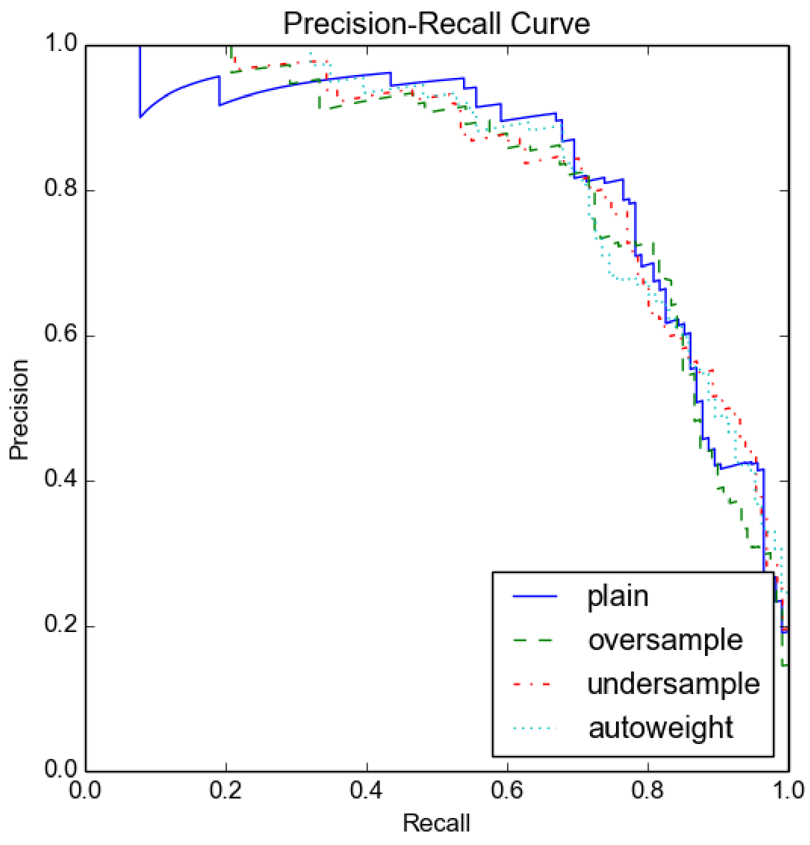


Figure 4.3: Precision-Recall Curves of Various Balancing Methods

of the according users were crawled from the present to the time of the mention, which allows for a more flexible analysis of the data.

4.4.1 Timelines and Message IDs

Each message sent on Twitter gets a unique ID. In September 2011, Twitter presented a new solution to assign IDs to tweets and direct messages, called Snowflake [Kin]. Before this, Twitter used globally sequential numbers, which turned out to be too difficult to synchronize. With snowflake, IDs are based on the creation time of the message as well as the ID of the snowflake worker client and a sequence number [twif]. Technically spoken, the IDs are now no longer fully sequential [Kin]:

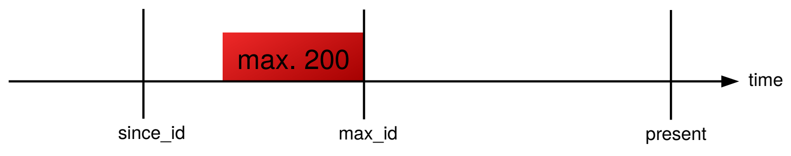


Figure 4.4: Effect of the max_id and since_id Parameters

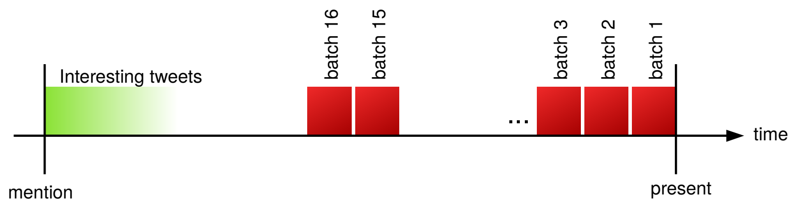


Figure 4.5: Twitter's API for Fetching Timeline Messages

> *In mathematical terms, although the tweets will no longer be sorted, they will be k-sorted. We're aiming to keep our k below 1 second, meaning that tweets posted within a second of one another will be within a second of one another in the id space too.*

4.4.2 users_timeline

The REST-API call to retrieve messages posted by a certain user is called `users_timeline`[1]. It fetches a maximum of 200 messages from a user specified by either the `user_id` or the `screen_name` field. Additionally, two values may be specified to further narrow down the area of interest: The `max_id` field limits the results to messages that have been posted before a certain ID, whereas the `since_id` selects tweets that have been posted after a certain ID. Within this range, Twitter will always return the most recent messages, as is shown in Figure 4.4.
As it is impossible to guess the correct `max_id` parameter, the only reliable approach would be to start from the most recent post (which is yielded by calling the API function with no parameters) and keep collecting until either 1.) the fetched messages are overlapping with the area of interest or 2.) the API doesn't allow to collect more tweets (only the 3,200 most recent tweets are available for applications to load). The scheme of this method is displayed in Figure 4.5.
Using this method restricts the crawling of user tweets, as many mentions used in the dataset were several years old at the time of processing.

[1]`https://dev.twitter.com/rest/reference/get/statuses/user_timeline`

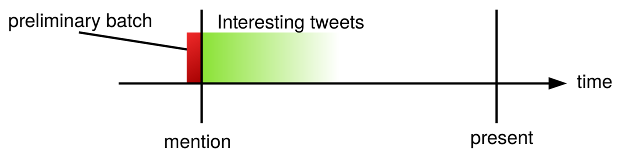


Figure 4.6: Preliminary Call to Prevent Unnecessary Message Fetching

Many users therefore already posted more than 3,200 messages since the mention in our dataset. So if the ID of the mentioned tweet is not greater than the oldest tweet recovered within 16 batches, the area of interest is not available for the application. Additionally, the sheer amount of users wouldn't allow an efficient crawling of the data in a decent amount of time for this thesis. Fetching messages from 100 random users took approximately 1 minute per user, taking Twitter's 180 requests per 15 minutes limitation into account. This would lead to a crawling time of 90 days for all 153,000 users in our dataset.

One measure to postpone this problem and get some data quickly is to sort the mentions chronologically and start crawling with the most recent ones, minimizing the chance of the mentioned user having posted too many messages since the mentioning tweet.

Another measure is to make one preliminary call with the `max_id` set to the ID of the mentioning tweet, as is shown in Figure 4.6. If Twitter answers to this request with a result, it means that the area of interest is available for the user to crawl. If no result is being returned, it means that the mention-ID is beyond the 3,200 tweet limit and the preceding message can not be fetched. This leaves a theoretical possibility of the preceding message exceeding the 3,200 tweet limit while the succeeding message would be within the fetchable range, which is ignored in this approach. The preliminary call does not yield any interesting tweets, as a possible result will only contain messages that have been sent before the mention happened. Therefore, the `count` parameter of this call can be set to 1 to minimize traffic and transmission time. This way, one can prevent crawling 16 batches of messages without having a result in the end.

Naturally, as a user can appear in multiple mention incidents that don't have to be at the same time, so prior to fetching the user's timeline, the mentions are sorted chronologically and the oldest mention is used for retrieving the appropriate time stamp.

Despite these counter-efforts it is clear that fetching the users response is a task that is optimally done at the same time as fetching the mention. An adaption to the crawler would eliminate the need of this step altogether, while offering a better answer of what actually happened to

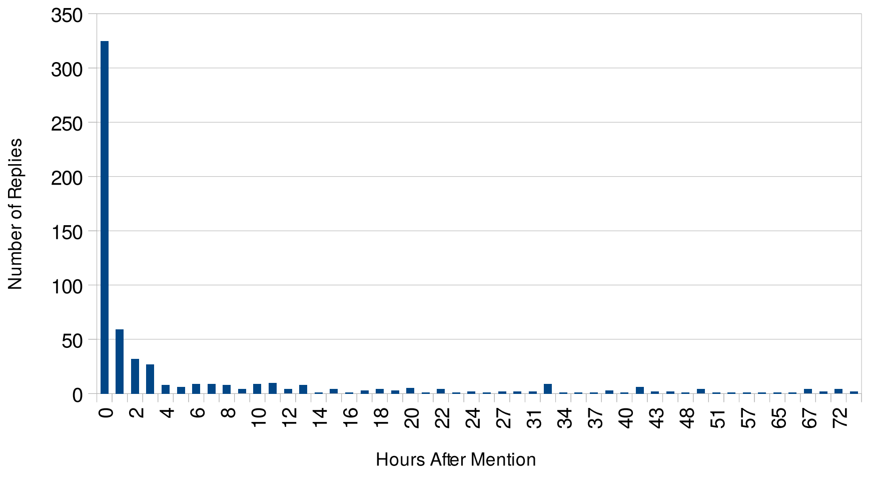


Figure 4.7: Duration Until Reply

the account: If the account is suspended either before or directly after the mention was posted the crawler could notice it, which suggests an actual hack.
With the problems described at hand, a set of 54,835 responses was crawled, where every response consists of multiple messages (theoretically up to 3,200).

4.5 Classifying Responses

Figure 4.7 shows the response time of the users from the manually classified messages. It can bee seen that most users that have responded did so within the first 24 hours after the mention, 48% of the replies happened within 60 minutes after the user received a hint.
Figure 4.8 shows a similar behavior regarding the number of messages that were sent before the corresponding user reacted. In 53% of the cases, the users reply with the first message they sent after they were mentioned. Of course, the user could have sent (possible spam) messages which were deleted later on and therefore were not fetched by the crawler.
Based on this data, the classification of the remaining mentions was limited to the responses that occurred within the first 10 messages, in which 95% of the users that were classified manually had responded. A mention and its following replies are classified as positive if the first 10 replies contain at least one positively classified response.
The classification itself is a bit more difficult in terms of what is seen as a positive match at all. Generally, one can distinguish following cases:

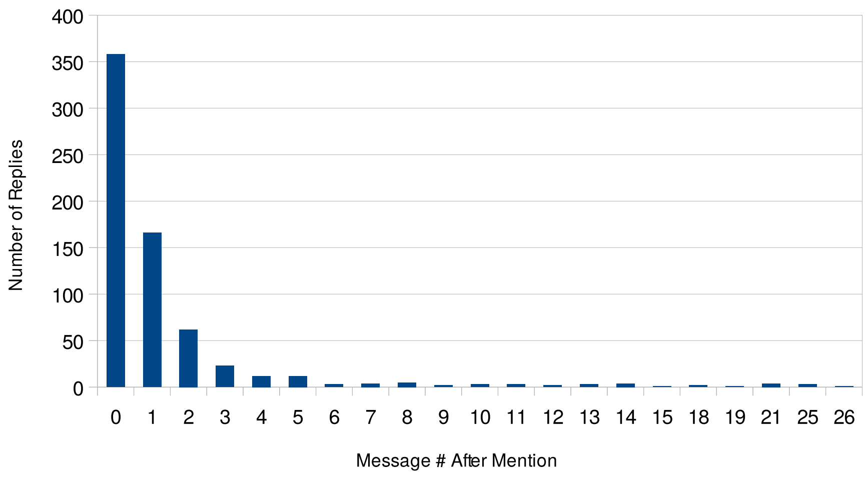


Figure 4.8: Message Number After Reply

1. The user did not respond in any way to the mentioning tweet.
2. The user responds to the tweet in a direct way.
3. The user responds in a general way to more than one tweet, not using a direct mention.

For the manual classification process, only two classes “positive” and “negative” were taken, for the sake of simplicity and better SVM performance. In this case, the “positive” class included the cases 2 and 3, which means that we consider a post a response even if the original writer didn’t send a direct mentioning tweet. More details on how the responses differ in classes can be seen in Chapter 6. The classification of the replies itself was performed in the same way than the first classification step and can bee seen in Figure 4.2. A set of 41,569 replies was classified manually, where 617 messages were classified as a positive response.
Figure 4.9 shows the precision vs recall curve of the second classification step using the same methods that were used in step 2. The according f1-scores can be seen in Table 4.5. The oversampling technique performs worse than in the previous classification, while the undersampling and autoweighting methods perform better. Taking the f1-score into account, the undersampling method was chosen for predicting the final response set.
This final classification step results in 16.452 responses classified as positive, which is 30% of the crawled set.

method	f1-score
plain	0.78
oversample	0.73
undersample	0.78
autoweight	0.70

Table 4.5: Precision vs. Recall Curve of Balancing Techniques on the Second Classification Step

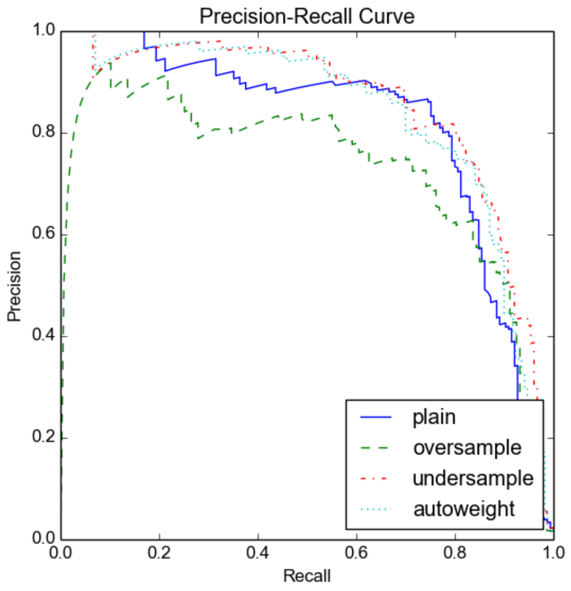


Figure 4.9: Precision vs. Recall of Second Classification

Chapter 5

Implementation

This chapter features detailed technical information about the implementation of the steps that are described in Chapter 4. It is sectioned according to the workflow described in Chapter 4.
The required parameters for all python scripts are described in Chapter 8.1.

5.1 Data Preparation

5.1.1 Sanitizing Json Files

The json-files provided by the Twitter API at the time of crawling did not have a valid json format when given. There were no separators between the single json objects, meaning the files mainly consisted of one single line which had up to 47,000 messages in them. To separate the tweets, the GNU tool `sed`[1] was used:

```
ls *.json | xargs sed -i 's/}{/}\n{/g'
```

This enabled further python scripts to read the tweets one by one by reading the files one line at a time. Another possibility would have been storing the tweets in a valid json array, but this was deliberately omitted because in many cases, the array would have been very large and parsing it would require a lot of main memory. This way, the lines can be read one by one, which reduces the memory consumption of the program, while not requiring notably more runtime. Some of the lines in the crawled files were empty, which were removed by following command:

```
ls *.json | xargs sed -i 's/^\w*$/g'
```

[1]`https://www.gnu.org/software/sed/`

5.1.2 Filtering Mentions

To remove the messages that don't contain any mentions, a python script `filter_mentions.py` was used.

The script checks the "entities" field of the json object in each line. If it is not empty, the messages is appended to the output file. Otherwise, it is ignored. In the output file, one message is stored per line.

The resulting json file is 7.3 GB large, which is difficult to handle. For convenience reasons, the file was split up in 1 GB large chunks using the GNU tool `split`:

```
    split -dC 1GB 01_mentions.json
```

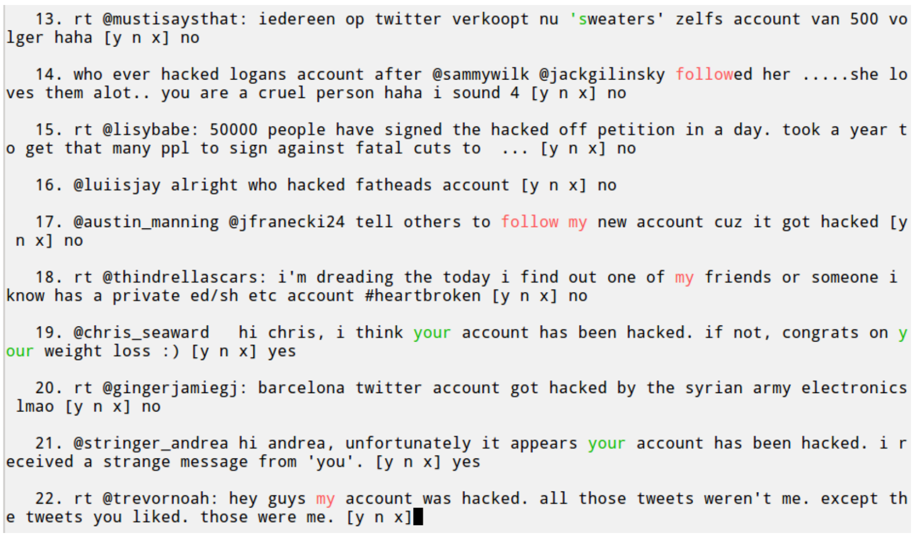
5.2 Extracting the Positive-Set

Before the SVM can classify the messages provided by step 1, a random subsample of tweets has to be classified manually. To extract a random subset of tweets, the script `01_select_random_tweets.py` is used.

Then, to manually classify these samples, the script `02_classify.py` can be used. It displays the content of each line read to the user, who has to type either "y" for a positive match, or "n", otherwise. To end the program, the user can type "x". The progress of classifying is saved, as the script detects how many lines the output file already contains and skips the first lines of the input file accordingly. To help the user detect the important parts of a message, certain keywords are highlighted in color, as can be seen in Figure 5.1. For example, the word *follow* often suggests a request to follow after an alleged hack of the own account and is often done to collect more followers (line 17) and does not suggest an other account to be hacked. The word *your* on the other hand is often important and suggests that the basic message involves an account belonging to somebody else (line 19). In this example, message number 19 and 21 are a positive match, while the others are negative.

After this step, the messages along with their according class have been saved to a json file. It is then used for training the SVM, which is done by the script `03_train.py`.

In this script, a TF-IDF vectorizer is used to extract features from the manually classified data, which then are used to train a linear SVM. The `scikit-learn` library offers a pipeline construct for classification, in which multiple elements can be concatenated.



```
   13. rt @mustisaysthat: iedereen op twitter verkoopt nu 'sweaters' zelfs account van 500 vo
lger haha [y n x] no

   14. who ever hacked logans account after @sammywilk @jackgilinsky followed her .....she lo
ves them alot.. you are a cruel person haha i sound 4 [y n x] no

   15. rt @lisybabe: 50000 people have signed the hacked off petition in a day. took a year t
o get that many ppl to sign against fatal cuts to  ... [y n x] no

   16. @luiisjay alright who hacked fatheads account [y n x] no

   17. @austin_manning @jfranecki24 tell others to follow my new account cuz it got hacked [y
 n x] no

   18. rt @thindrellascars: i'm dreading the today i find out one of my friends or someone i
know has a private ed/sh etc account #heartbroken [y n x] no

   19. @chris_seaward   hi chris, i think your account has been hacked. if not, congrats on y
our weight loss :) [y n x] yes

   20. rt @gingerjamiegj: barcelona twitter account got hacked by the syrian army electronics
 lmao [y n x] no

   21. @stringer_andrea hi andrea, unfortunately it appears your account has been hacked. i r
eceived a strange message from 'you'. [y n x] yes

   22. rt @trevornoah: hey guys my account was hacked. all those tweets weren't me. except th
e tweets you liked. those were me. [y n x]
```

Figure 5.1: Manual Classification Process

```python
pipe = Pipeline([('tfidf', TfidfVectorizer()),
                 ('ch2', SelectKBest(chi2)),
                 ('clf', LinearSVC())])
crossvalidation = 5
parameters = {
    'tfidf__ngram_range': [(1, 2), (1, 3), (2, 3)],
    'tfidf__stop_words': [None, 'english'],
    'tfidf__max_df': [0.5, 0.75],
    'tfidf__min_df': [0.0001, 0.001],
    'tfidf__analyzer': ['word', stemming_analyzer,
                        lemming_analyzer],
    'tfidf__norm': ['l1', 'l2'],

    'ch2__k': [200, 'all'],

    'clf__C': [0.1, 1, 10],
}

gridSearch = GridSearchCV(
    pipe,
    parameters,
    n_jobs=-1,
    scoring="f1",
    cv=crossvalidation
)
```

Listing 5.1: Example of Possible GridSearch Parameters

In Listing 5.1, such a pipeline is constructed, featuring a TF-IDF vectorizer, a select-k-best filter using a χ^2 function, and a linear SVM classifier. It is mainly for user's convenience, as the data can be trained to the whole pipeline, which then performs feature extraction, filtering and training in one step. The parameters of the vectorizer, the χ^2 filter and the SVM are optimized automatically by the `scikit` featured `GridSearch`. It accepts a range of possible values and creates every possible combination. An example of possible parameters for the vectorizer and SVM can be seen in Listing 5.1 on lines 5 to 17. For this example, the GridSearch module would try 1156 combinations. It is notable that the parameter set contains possible parameters for the vectorizer, the χ^2 filter and the SVM at one single location, making full use of the pipeline construct. The actual parameters that were tested and their optimal result are displayed in Table 5.1. Additional parameters used by the GridSearch module include a number of threads that the program should use (line 22, where -1 stands for "use all processor cores available"), which measure should be optimized (line 23) and how many cross validation steps should be performed (line 24).

Parameter	Description	Optimal Value
	TF-IDF Vectorizer	
ngram_range	specifies which n-grams the vectorizer should consider. This parameter consists of a lower and upper bound. For example, (1,4) means that the vectorizer would consider all n-grams of size 1 to 4.	1-3
stop_words	`scikit` features a built-in list of English stop words, which can be removed before feature extraction.	none
max_df	if specified, the vectorizer ignores terms that have a document frequency that is higher than this value.	0.9
min_df	if specified, the vectorizer ignores terms that have a document frequency that is lower than this value.	0.001
use_idf	enable weighing by idf in general.	True
strip_accents	if set, removes accents from characters (e.g. transforms à into a) according to the encoding specified	unicode
analyzer	specifies what the vectorizer considers a term (e.g. words or characters). A method can be supplied to generate the n-grams manually.	lemmatizing_analyzer
norm	used to normalize the term vectors. Can either be 'l1', 'l2' or None.	l2
	χ^2-filter	
k	How many features the filter should select, or `all` to use all features.	all
	SVM	
C	The penalty parameter for the error term. See Chapter 2 for more details.	1.0

Table 5.1: Optimal Parameters Found by GridSearch

As is described in Section 2.2.5, multiple methods were tested to cope with the imbalance of the training set. The built-in grid search engine of `scikit` does not feature a specific sampling parameter, but rather allows the user to specify custom cross validation patterns. This feature was used to build a sampling stratified K-fold cross validator, which can either under- or oversample the training parts of each fold. The oversampling itself was done by randomly copying samples of the under-represented class until the class sizes were equal. For undersampling, a random subset of the overrepresented class with the size of the underrepresented class was taken for training. Since the cross validation pattern is a parameter of the grid search itself (rather than one of the pipeline elements) and cannot be set to multiple values, the whole grid search process was performed multiple times, once for each method. The total amount of grid search training runs equals

$$m_{unb} \cdot k_{cv} \cdot \prod_{i \; in \; params} |i|$$

where m_{unb} denotes the number of methods used to cope with the unbalanced training set, k_{cv} are the amount of folds being calculated for cross validation and i denotes a set of possible values for a specific parameter and $|i|$ its length. For $k_{unb} = 4, k_{cv} = 5$ and the parameters as explained in Table 5.1, this totals to a 138,240 training and testing runs.

5.2.1 Including Linguistic Preprocessing

Several linguistic optimizations have been tested for suitability in the prediction step. For the lemmatizing functionality, the python library `spaCy` is used. Within the workflow, it is used by the vectorizer as a pre-processing step. On line 10 of Listing 5.1 the function `lemming_analyzer` is added to the grid search parameters. This way, it can be automatically tested whether the SVM actually profits from the lemmatizing without having to write own test cases.
Similarly, stemming is also included at this point. As it is not useful to combine stemming and lemmatizing, the two options are mutually exclusive token analyzers that are tested within the grid search. For stemming, the Porter Stemmer algorithm is used by the `nltk` natural language toolkit library[2].
The domain-specific optimizations of hashtag-, url- and mention-removal are not included in the grid search parameters as this was technically not possible in an easy way. Instead, these measures were tested after having determined the optimal values for the remaining parameters by trying all possible combinations. By design, this leaves the possibility

[2] `http://www.nltk.org/`

of a specific combination of the parameters together with the removal of one of those categories undetected. However, the results of these measurements, which are listed in Chapter 7, show that they hardly influence the classifier at all. This suggests that the possibility of a better combination existing at all as well as the difference to the best found parameter set is small.
Having detected the optimal classifier for the problem at hand, the script `04_predict.py` can be used to predict the classes of the remaining messages in the *Mention*-set.

5.3 Removing Inactive Users

The next step is to remove the users from the dataset that are no longer available on Twitter, either because their accounts have been deleted or deactivated. In theory, this step could have been merged with the next one, but it is more efficient to check for the users separately. The Twitter API call for looking up user names allows to search for 200 names at once, making it far more effective.
For the Twitter API calls, the library `python-twitter`[3] has been used. The script automatically sleeps for 5 minutes if the Twitter's rate limit has been exceeded. A typical sample output is shown in Listing 5.2.

```
    [DD] batch 539 of 3511: returned 92 names
    [DD] batch 540 of 3511: returned 92 names
    [NN] rate limit exceeded, sleeping for 5 minutes
    [NN] rate limit exceeded, sleeping for 5 minutes
    [NN] rate limit exceeded, sleeping for 5 minutes
    [DD] batch 541 of 3511: returned 88 names
    [DD] batch 542 of 3511: returned 88 names
```

Listing 5.2: Sample Output of the User Existance Script

The script also reorders the structure of the output json file. Until this step, each json file consisted of one tweet per line, encoded in a json object. Since a single user can be mentioned in multiple messages or be the subject of a retweet, a logical structure of the data is to group the messages according to the users mentioned.

```
{
    'uid': 123456789,
    'mentions': [ {id: ...}, {id: ...}, ... ],
    'timeline': [],
}, ...
```

On the other hand, this induces a multiplication of messages that contain multiple mentions. Figure 5.2 displays the distribution of mention

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

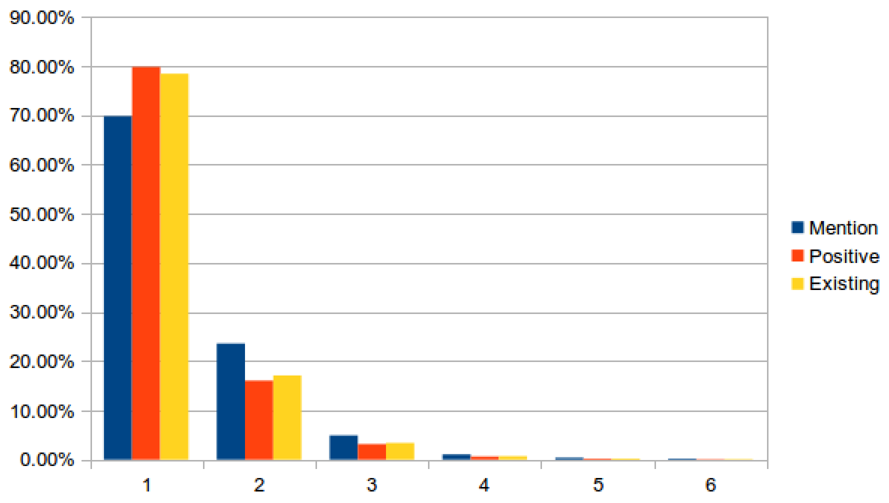


Figure 5.2: Ratios of Multiple Mentions Among the Message Sets

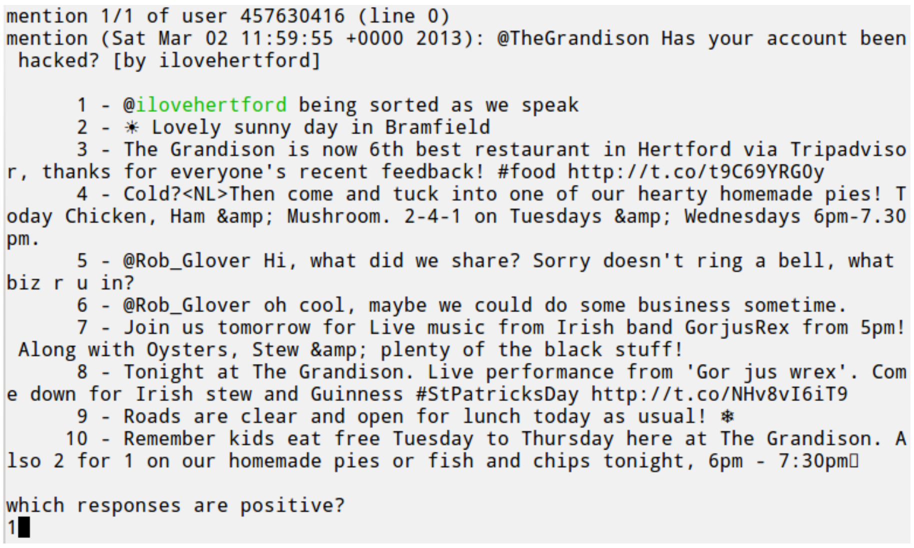
amounts in the *positive*-set and shows that the majority of the messages contain only one mention, making this problem less relevant.

5.4 Fetching Responses

The script `01_fetch_timelines.py` takes the data from the previous step and tries to collect the responses by the allegedly hacked users.
To increase the robustness of the fetching process, an additional store for saving already handled user names was implemented. For this purpose, a document based database MongoDB[4] was set up which only contains the user ids of already handled mentions. This way, the script can be arbitrarily started and stopped without losing the progress. Using MongoDB for the entire message set turned out to be too slow.
In detail, following steps are executed:

1. The mentions for each user are sorted chronologically.
2. The MongoDB is checked whether the user was already handled or not. If so, the next entry is handled.
3. Starting with the oldest mention, a preliminary call is made according to Section 4.4 to check whether the response for this mention is available.
4. If the result of step 2 is True, all tweets of this user until said mention are crawled using the API. This way, all possible responses to newer mentions are covered as well.

[4] `https://www.mongodb.org/`



```
mention 1/1 of user 457630416 (line 0)
mention (Sat Mar 02 11:59:55 +0000 2013): @TheGrandison Has your account been
 hacked? [by ilovehertford]

      1 - @ilovehertford being sorted as we speak
      2 - ☀ Lovely sunny day in Bramfield
      3 - The Grandison is now 6th best restaurant in Hertford via Tripadviso
r, thanks for everyone's recent feedback! #food http://t.co/t9C69YRG0y
      4 - Cold?<NL>Then come and tuck into one of our hearty homemade pies! T
oday Chicken, Ham &amp; Mushroom. 2-4-1 on Tuesdays &amp; Wednesdays 6pm-7.30
pm.
      5 - @Rob_Glover Hi, what did we share? Sorry doesn't ring a bell, what
biz r u in?
      6 - @Rob_Glover oh cool, maybe we could do some business sometime.
      7 - Join us tomorrow for Live music from Irish band GorjusRex from 5pm!
 Along with Oysters, Stew &amp; plenty of the black stuff!
      8 - Tonight at The Grandison. Live performance from 'Gor jus wrex'. Com
e down for Irish stew and Guinness #StPatricksDay http://t.co/NHv8vI6iT9
      9 - Roads are clear and open for lunch today as usual! ❄
     10 - Remember kids eat free Tuesday to Thursday here at The Grandison. A
lso 2 for 1 on our homemade pies or fish and chips tonight, 6pm - 7:30pm

which responses are positive?
1
```

Figure 5.3: Example of Classifying Responses

After having fetched all data for this user, the timeline is cropped to contain only 10 messages after each mention, which is mainly done to save disk space. As described in Section 4.4, this still covers the majority of relevant response messages.
The script stores the data in the same style as the previous step, while filling the *timeline* field of the map. It takes a very long time, as is described in Section 4.4.

5.5 Classifying Responses

The classification of the responses is done similar to Section 5.2. First, a random subset of messages was taken and manually classified. For this, a script `01_classify.py` is available, which displays the mention and the according replies comfortably to the user. In contrast to the first classification step, the input of this script is already a single file, so the script itself takes care of selecting the messages randomly.
Figure 5.3 shows an example of the manual classification process. The user can then enter the numbers of the messages that represent a positive response, separated by commas. Important terms, such as the username of the mentioning author are highlighted in green for easier recognition. All messages that are entered by the user are marked as positive, while all others are marked negative. The script outputs the data in the simple format that stores the tweet text and its class in a json array. One array is stored per line.
The script `02_train.py` is subsequently used for training a SVM with

the manually classified data. Similar to the previous classification step, it also uses four different approaches to deal with the class imbalance. The optimal parameters that were determined by grid search are exactly the same as in the previous classification step and thus are shown in Table 5.1.
The final step was to predict the remaining responses of the alleged hacker victims. The script `03_predict.py` predicts the classes of these answers and outputs some statistics about the data set.

Chapter 6

Qualitative Analysis

This chapter describes the variety of different messages that are handled within this project. All messages written in *italics* are actual messages crawled from Twitter. Newline characters have been removed for visual purposes. In the remainder of this chapter, the author of a message containing a mention will be denoted as "A: ", whereas the alleged victim is displayed as "B: ". Some conversations are described in detail, explaining the background context of them.

6.1 Initial Data

The origin of all analyzed messages is a set of tweets that was crawled by the DBIS institute. The crawler used reads the public Twitter stream (see Section 2.1.8), filtered by the word "hacked". This results in a very broad set of messages that can mainly be split into three groups:

1. Messages we are actually looking for, suggesting that other Twitter accounts have been hacked

    @AllRiseSilver is your twitter account hacked?

2. Messages where the author of the suggestion is the alleged victim

    Looks like my twitter account got hacked. I didn't lose 2.5lbs

3. Tweets about other accounts that have been hacked (email, banking), regardless of the owner

    @AmpersUK Looks like you Gmail account hasbeen hacked - sending requests for money to be lent to you on holiday in the Ukraine!

The latter two groups are of no interest for this thesis, but the first one is worth having a closer look at. Messages that do not contain any mentions mostly do not have the purpose of informing said person of any hacking events, as this makes it very unlikely for them to read the tweet. These messages include stories that are shared amongst friends:

> *So my stepsister's account has been hacked and I automatically get the blame. wow gee thanks.*

Also theses tweets are of limited usefulness for the thesis, as without any mention, there is no way to perform automated analysis of a possible response.

6.2 Celebrities on Twitter

As shown in Table 4.2, many messages on Twitter are about famous singers. This is often abused by users that try to collect as many followers as they can:

> *@Calum5SOS hi cal,I hope youre having fun in America!! If you happen to see this pretty please refollow me,someone hacked you on my account*

These type of messages are often flooded and reposted over the network. Sometimes, celebrities' Twitter accounts get hacked, often resulting in tweets without mentions, as these messages are not really trying to inform the hacked account of anything:

> *Eunhyuk's Twitter account got hacked? That's some scary shit right there....*

The big amount of retweets of these messages cause mentions of celebrities to be amongst the top, also in the *positive*-set.

6.3 Relevant Suggestions

Having a closer look at the messages that are of interest, different groups can be distinguished as well.

1. Messages that plainly suggest a hack.

   - *@AllRiseSilver is your twitter account hacked?*

2. Suggestions to take a specific counter action. In many cases users suggested the victims should change their password:

   - *@aam429 I think your account has been hacked change your password good luck*

   Other measures include looking into the Twitter authenticated applications or search for other suitable measures:

- *@brijesh58 Did you DM me any link ? Or is your account hacked ? Check apps you have granted permission.*
- *@barreltopwagons change password quick! Google twitter account hacked - very helpful !*

3. Retweets of the alleged spam

   - *Bwahahaha! Is your account got hacked bro? RT @owlcity: j0mbl0 h4h4h4 lu k3n4 v12u5 4l4y y4? k37ul424n cy4ph4 wkwkwk -_____-*

4. Some describe the source of the spam in detail. Often, the hacked account does not send tweets but instead (or additionally) sends direct messages

   - *@BeckyBeckyh123 I think your account has been hacked, just received a spurious DM from you*

5. messages referring to the content of the spam

   - *@jessicalacie I think you've been hacked, got a dieting DM from your account.*
   - *Hey @djonesjax , either you are using amphetamines or someone has hacked your account*

6.4 Responses

Having the according responses to above mentions available, small conversations can be analyzed. In general, the first and most obvious discrimination can be made whether the mentioned user responded at all. A user not responding to a mention can have multiple causes, like not having regained control over the account:

A: *@terrileonard88 you keep tweeting about loosing weight ? Is your account hacked ?*
B: *burn the fat off while removing 2 inches or more from your tummy with http://t.co/dYiXiCSob6*
- *Quickly burn off stomach fat while dropping 25lbs in a month using http://t.co/BWn4JYPYWB*
- *Cut mass from your tummy and drop 21lbs of weight in 30days http://t.co/y52tZS8Iy4*
- *...*

But other scenarios are more vague. Often, the victim sends valid messages after the mention and does not respond to anything, leaving the question of what actually happened. A user might have responded to a message but then deleted the conversation, to conceal the hacking event or just used other communication channels than Twitter for the incident. Positive responses that confirm the hacking of an account also appear in a variety of classes. The most straight forward response is a tweet directly answering the mention:

A: *@juzz_hot Hey, think ur account is hacked. Have like 3 DMs from u within the last 30mins. Change ur password*
B: *@Alisha_Salik thanks dude! Will do*

In many cases, the victim additionally apologized for sending spam by posting a general tweet rather than responding to a specific mention:

> *sorry for any recently sent spam messages - my twitter account was hacked...*

Some users also address the content of the spam their account was sending or mentioning that they actually changed their password:

> *@Hooderman , think I got it changed. sure didn't loose 20 pounds. Thanks*

Even a possible distribution of malware might be uncovered using responses:

A: *@Castricato wtf your account is hacked!*
B: *@doomilicious lol ya yours too I got a private message from u :p and 30 other ppl on my twitter account*

And sometimes the struggle of regaining control can be seen in the conversation:

A: *@GraJay Please attend to your (hacked) Twitter account. :-)*
B: *My account has been hacked - trying to sort- apologies*
- *Lose 20lbs of body-fat in less than 2 weeks http://t.co/zq0axTbp19*
- *Lose 20lbs of body-fat in less than 2 weeks http://t.co/jez8iXudb4*
- *Lose 20lbs of body-fat in less than 2 weeks http://t.co/nxcupDt9NM*
- *Think hacked account now sorted-thanks for messages*

Chapter 7

Results and Discussion

Of the 54,835 mentions that were analyzed, the second classification step predicts a total of 16,452 messages to have a positive response. This means that 30% of the users that were mentioned in a message suggesting a hack have replied. These responses include both negative and positive responses, which are not distinguished in this approach.

This number suggests that it is highly feasible to analyze not only the hacked account itself, but also its peers to detect possible events. This approach also overcomes previous limitations that content based analyzers are prone to, such as spam distribution over direct messages. Many posts mention the victim sending messages over DMs, which are not publicly visible and therefore cannot be analyzed by any API calls.

The results of analyzing the average response times in Figures 4.7 and 4.8 suggest that most users already react quickly to messages mentioning them. This short time shows that it is important for most users to clarify what is going on.

7.1 Effect of Lemmatization

Since lemmatizing the input was selected by the grid search to be preferable, this improvement has been analyzed in detail. Figure 7.1 shows the effect of lemmatizing on the PR-curve. It can be seen that though time consuming (disabling it improved the runtime by 47%), is very effective in increasing the performance of the SVM. Naturally, a possible countermeasure for the processing overhead would be to calculate the lemmatized texts of each message beforehand.

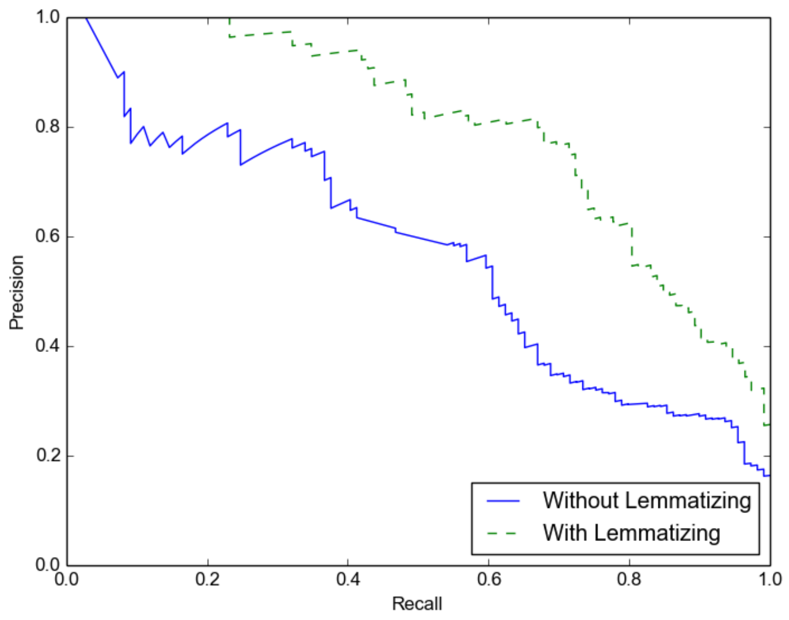


Figure 7.1: Effect of Lemmatizing on PR-Curves

7.2 Stop Word, Hashtag, Mention and URL Removal

Intuitively, stop words do not add any specific information to a text and their position is also irrelevant when using a TF-IDF vectorizer. The same arguments hold for mentions and URLs, as they are not likely to appear in relevant posts very often. However, the grid search did not select the removal of either of these features to be beneficial for the classification process. Table 7.1 displays the effect of removing hashtags, mentions and URLs from messages before training the SVM. The numbers represent the average of 100 repetitions to minimize deviation. The data suggests that removing the mentioned fields does not have a great effect on the performance of the classifier, which is surprising. However, for some of these fields explanations can be found:

- Mentions: When manually classifying the messages, mentions of celebrities often are a clear indicator of the message being non-relevant (see Section 6.2). Considering that these mentions are common (see Table 4.2), this might actually help the classifier.

- URLs. Considering that URLs are mostly unique, there is a high chance of a URL being ignored by the TF-IDF vectorizer, which

Class	Hashtag	Mention	URL	f1	recall
1				0.760	0.741
2			✓	0.759	**0.742**
3		✓		0.758	0.736
4		✓	✓	0.753	0.738
5	✓			**0.761**	0.735
6	✓		✓	0.759	0.736
7	✓	✓		0.757	0.741
8	✓	✓	✓	0.758	0.741

Table 7.1: Effect of Preprocessing and Removing Hastags, Mentions and URLs on the F1 and Recall Score. A Tick in the Table Means the According Feature was Removed From the Text.

was set to ignore terms occurring less than a specific value. Details on this parameter are explained in Table 5.1.

Counter-intuitively, removing hashtags is not having any effect on the classification, which cannot be explained at this point. From the 2,295,831 messages of the *mention*-set only 219,933 contain hashtags (9.58%), which might indicate that there are just too few to have an impact on the classifier.

7.3 Limitations

Due to the complexity of natural languages, it is not always clear whether a response was the direct result of a mention. In many cases users respond by apologizing to a larger audience than a single person that mentioned a possible hack. Of course, this does not infer how the user was informed about the event itself. Users may have detected the hack by themselves or by other communication channels.
An important measure that could not be determined due to the Twitter limitations is the fraction of users that reply to the mention in general. We analyzed a set of responses and conclude that 30% of the people that *we had access to* did respond in a positive way. No conclusions can be made on how many did respond in total, as the Twitter API restricts access to old messages.
One problem that occurs often in Twitter data analysis is the influence of celebrity user accounts. In the dataset that was analyzed, few messages were retweeted very often. The influence of these few tweets cannot be estimated at this point and requires further investigation.

Chapter 8

Conclusion and Future Work

Using machine learning methods, we have shown that a significant amount of users respond to suggestions of their accounts being hacked. For this, a TF-IDF feature vectorizer was used together with linear Support Vector Machines, and several linguistic improvements were tested for suitability. A brute force grid search algorithm determined the best parameters for the classification step.
Additionally, a qualitative analysis shows the variety of different conversation types that have been analyzed.
The results gathered in this research can be used as additional information when trying to detect hacked accounts. Many previous approaches already focus the social environment of Twitter users, but only regarding the statistics of the follower/followee relationship. Using the content of messages that suggest hacks could possibly enhance existing methods, as it overcomes obstacles that can't be solved by just analyzing the account itself.
In general, this method could be useful to be implemented along other methods in order to build or complete more complex spam detection systems. Using the results described above, several key features may be improved in further implementations:

- Gather data in real-time. The greatest bottleneck in data analysis is the reduced amount of tweets that can be gathered after some time has passed. By implementing a real-time version, the responses of the alleged victim can be analyzed directly by using Twitter's user-streams. This prevents data loss can answer additional questions, like: Does the user post messages at all or is the account being suspended at some point? If the user does respond, is the content legit or actual malicious content? If the user responds to mentions, what message is being delivered? These questions are being answered qualitatively in this thesis, but could also be answered quantitatively for a suitable dataset. Since the machine learning part of this approach can be configured beforehand, a real-time analyzer could be set up easily.

- Use additional information. The presented approach relies only on the content of the messages themselves. Additional Information, like analyzing the user relationship between the mentioned user and the original author or the total amount of followers/followees may be of interest (e.g. for detecting celebrity user accounts).
- Optimize the machine learning part in the prediction process. This thesis used SVMs to predict message classes, but other methods exist. Also, complex approaches for generating artificial messages for oversampling like SMOTE may be of use.

Together with existing methods that analyze the allegedly hacked user account itself, a combined detection system could easily be realized.

Appendix

A.1 Required Parameters for Python Scripts

Step 1 - Data preparation

`filter_mentions.py`	
-i DIR	directory containing the sanitized .json files
-o FILE	output file to store the *mention*-set

Step 2 - Extracting the Positive-Set

`01_select_random_tweets.py`	
-i FILE	input json file
-o FILE	output json file containing the random messages
-n INT	number of messages to randomly select

`02_classify.py`	
-i FILE	input json file of randomly selected messages
-o FILE	output file with classified messages

`03_train.py`	
-i FILE	a json file containing manually classified messages
-o FILE	the output file where to store the optimal SVM parameters

`04_predict.py`	
-p FILE	The file containing the optimal classifier found by `03_train.py`
-o FILE	The output json file containing the *Positive*-Set
FILE...	The input json files

Step 3 - Filtering for Existing Users

`01_filter_existing_users.py`	
-op FILE	File location of the output json file that contains the still existing and available users and their tweets
-on FILE	File location of a list of user names that is no longer active
FILE...	Input json files

Step 4 - Fetching User Timelines

`01_fetch_timelines.py`	
-i FILE	input json file that contains grouped mentions
-o FILE	output json file containing the timelines

Step 5 - Classifying Responses

`01_classify.py`	
-i FILE	Input json file that contains one user dictionary per line.
-o FILE	Output json file.

`02_train.py`	
-i FILE	Input json file from the manual classification step
-o FILE	Output file that stores the optimal found parameters for the SVM

`03_predict.py`	
-i FILE	Input json file from the manual classification step
-p FILE	The file containing the optimal found parameters by the `02_train.py` script
-o FILE	Output file that stores the optimal found parameters for the SVM

Bibliography

[abo] About twitter. https://about.twitter.com/company. Accessed: 09.06.2015.

[AC$^+$10] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[BGL10] Danah Boyd, Scott Golder, and Gilad Lotan. Tweet, tweet, retweet: Conversational aspects of retweeting on twitter. In *43rd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10. IEEE, 2010.

[BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BMRA10] Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgilio Almeida. Detecting spammers on twitter. In *Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference (CEAS)*, volume 6, page 12, 2010.

[Bry] Martin Bryant. Twitter to cut off firehose resellers as it brings data access fully in-house. http://thenextweb.com/dd/2015/04/11/twitter-cuts-off-firehose-resellers-as-it-brings-data-access-fully-in-house. Accessed: 18.02.2016.

[DPHS98] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155. ACM, 1998.

[ESKV13] Manuel Egele, Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Compa: Detecting compromised accounts on social networks. In *NDSS*, San Diego, CA, USA, 2013.

[Fis36] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[GTPZ10] Chris Grier, Kurt Thomas, Vern Paxson, and Michael Zhang. @spam: The underground on 140 characters or less. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 27–37, New York, NY, USA, 2010. ACM.

[HCL03] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.

[Hos05] Véronique Hoste. *Optimization Issues in Machine Learning of Coreference Resolution*. PhD thesis, Universiteit Antwerpen. Faculteit Letteren en Wijsbegeerte, 2005.

[Hue] Roman Huet. Connecting to the pulse of the planet with twitter apis. `https://blog.twitter.com/2014/connecting-to-the-pulse-of-the-planet-with-twitter-apis`. Accessed: 04.11.2015.

[Joa98] Thorsten Joachims. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. Springer, 1998.

[Kin] Ryan King. Announcing snowflake. `https://blog.twitter.com/2010/announcing-snowflake`. Accessed: 02.07.2015.

[KKL$^+$08] Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M Voelker, Vern Paxson, and Stefan Savage. Spamalytics: An empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 3–14. ACM, 2008.

[KM97] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.

[Leh] Christian Lehmann. Lemmas and lemmatization. `http://www.christianlehmann.eu/ling/ling_meth/ling_description/lexicography/lemmatization.html`. Accessed: 15.02.2016.

[MC11] Michael McCord and M Chuah. Spam detection on twitter using traditional classifiers. In *Autonomic and Trusted Computing*, pages 175–186. Springer, 2011.

[Mes] Jana Messerschmidt. Twitter welcomes gnip to the flock. `https://blog.twitter.com/2014/twitter-welcomes-gnip-to-the-flock`. Accessed: 18.02.2016.

[Por80] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[sci] Scikit-learn documentation. `http://scikit-learn.org/stable/documentation.html`. Accessed: 19.02.2016.

[SKV10] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 1–9. ACM, 2010.

[SWY75] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, Nov 1975.

[tco] Link sharing made simple. `https://blog.twitter.com/2011/link-sharing-made-simple`. Accessed: 26.01.2016.

[TGSP11] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. Suspended accounts in retrospect: An analysis of twitter spam. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 243–258. ACM, 2011.

[twia] `https://support.twitter.com/articles/31796-my-account-has-been-compromised`. Accessed: 26.01.2016.

[twib] Bit.ly eclipses tinyurl on twitter. `http://bits.blogs.nytimes.com/2009/05/07/bitly-eclipses-tinyurl-on-twitter`. Accessed: 02.12.2015.

[twic] Posting links in a tweet. `https://support.twitter.com/articles/78124`. Accessed: 02.12.2015.

[twid] Removing the 140-character limit from direct messages. `https://blog.twitter.com/2015/removing-the-140-character-limit-from-direct-messages`. Accessed: 02.12.2015.

[twie] Twitter filter api. `https://dev.twitter.com/streaming/reference/post/statuses/filter`. Accessed: 15.02.2016.

[twif] Twitter ids. `https://dev.twitter.com/overview/api/twitter-ids-json-and-snowflake`. Accessed: 19.02.2016.

[twig] The twitter rules. `https://support.twitter.com/articles/18311`. Accessed: 26.01.2016.

[Vap95] Vladimir N Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[YP97] Yiming Yang and Jan O Pedersen. A comparative study on feature selection in text categorization. In *International Conference on Machine Learning (ICML)*, volume 14, pages 412–420, 1997.

[ZS14] Eva Zangerle and Günther Specht. ”Sorry, I was hacked” - A Classification of Compromised Twitter Accounts. In *Proceedings of the 29th ACM Symposium on Applied Computing*, pages 587–593, Gyeongju, Korea, 2014. ACM.