Utrecht University

Master's thesis

Artificial Intelligence

Text Classification of Dutch police records

Author:
Minke Brandenburg

First supervisor:
prof. dr. A.P.J.M. Siebes
Second supervisor:
dr. A.J. Feelders

July 2017

Utrecht University

Contents

Chapter 1

Introduction

In all facets of modern life data is generated. Most of our choices and actions are documented, both in obvious and less obvious ways. All this data contains a lot of information about us and the world around us. To disclose this information, data mining algorithms can be used. machine learning and data mining have been a hot topic among businesses and governments. Machine learning algorithms take large amounts of data, and find the patterns that it contains that might be too subtle for human eyes to see. The aim of these algorithms is to discover the specific patterns that can be found in large amounts of data.

Because of these large amounts of data, text mining, as a specific branch of data mining, is the process of extracting interesting information from large bodies of text data. Examples are text classification, sentiment analysis, topic mining and text summarization. This thesis focuses on text classification. Text classification or text categorization is the task of labelling text documents. Depending on the specific task, each document can have one or multiple labels that depend on the content of the document. In text classification the goal is to create a model using training data that, given new data, can classify the new documents correctly.

1.1 Problem description

The large databases of government agencies are an interesting source for analyzing. Using government data, tax evaders might be found, and people could be called in for medical testing more or less often based on their data. This thesis project, part of a project for the the WODC (Research and Documentation Centre) of the Dutch Ministry of Security and Justice, is an effort to use text mining for the classification of police reports.

Police reports and statements are written by police officers all the time. According to a 2016 report from the Dutch Center for Statistics, just in 2015 there were 840.340 victim reports and statements entered into the Dutch police registration database (Basisvoorziening Handhaving, BHV)[1]. While it is possible to enter a topic or category, this is sometimes forgotten or ignored. Other times a report belongs to multiple categories, where only one might be given. These reports contain a lot of text. To go through all of these by hand to label them would require immense amounts of time and effort. However, having labeled reports is very practical for multiple reasons. It greatly improves the ease and accuracy with which the yearly crime numbers are calculated for different categories of crime, and can make it easier to spot at once with what category of crime a person is associated

[1]Source: CBS, Slachtofferregistraties bij de politie en Slachtofferhulp Nederland, 2010-2015

in the system.

This research project focuses on the latter. The goal is to create a model that could classify several categories of online crime. This would enable researchers to calculate which people in the system were associated with multiple different cases in a period of time, and thereby calculate how many people recidivate on those categories of online crime. The research question of this thesis therefore is:

“How can we use text mining techniques to correctly classify police data?”

In the rest of this thesis I will give a short overview of some of the most common steps and algorithms in text classification tasks. In chapter 2.1 I will discuss the data set used in this specific project. The chapter that follows is more general in nature, and discusses important steps in preprocessing (section 3.1) and feature selection (section 3.2). After that I will discuss some common supervised and semi-supervised machine learning algorithms in section 3.5. The methods discussed in these sections were all used in or considered for the experiment. The experiment and the results are discussed in chapter 4, with some discussion and suggestions for future research in chapter 5.

Chapter 2

Data set

2.1 Data

The data set used in this thesis is a set of police reports from the Dutch police. It includes statements from both witnesses and suspects, and observations made by police officers on the scene or during interviews. For each report the following data was available: report registration ID, the police-unit responsible for the report, and ID, sex, place of residence, nationality and age of the suspect, and the text of the report itself. In this project only the text was used, although the results can be combined with the other data to possibly improve the results. The reports are from the years 2013, 2014 and 2015, and were preselected by a very broad query on the police database[1]. This query can be found in appendix A, and consists of a set of terms that may imply some relation to online crime.

The full data set contains over 242.987 unique[2] records, all written in Dutch. Since a preliminary report was needed about halfway during the allotted time for the project, the research was divided in two phases. For the first phase of this projects, a small subset of 1896 of these records was labeled by two annotators. These labeled records were evenly spread over the time period 2013-2015. The records to be labeled were selected in smaller subsets of between 100 and 200 records, by a semi-randomized selection algorithm. The query in appendix A that was used to retrieve the data from the police database selected records if they contained words from at least one of three categories. Since much more records fell in one of the three categories than in the other two, the selection for annotation was done by randomly selecting $^1/_3$ of the set for annotation from each of these categories. So to clarify: each part of the query yielded 632 records for annotation. The goal of this selection process was to try and keep the categories balanced as much as possible.

To annotate the records, the annotators established whether the incident described in a report belonged to one or more of three categories: online threats, online distribution of sexually obscene images, and computer trespassing. The criteria used for the annotation are shown in table 2.1. To ascertain that both annotators used the same criteria, 98 records of the set were annotated by both annotators. In 5 cases they disagreed over the categorization of a record, of which 2 disagreements were about online threats and 4 about computer trespassing[3]. For these cases only the negative categorization [4] was taken into account for the remainder of the project, to avoid ambiguity in the

[1]Basis Voorziening Handhaving (BVH) of Dutch National Police

[2]Some records appear multiple times, for example once for every suspect in a case. In these cases the extra records were filtered out.

[3]1 record belonged to both categories

[4]Not belonging to the interesting class

Online threats
Threats issued through online means On a smartphone: traditional text messages or calls are not seen as online, while apps (Whatsapp, Skype, Facebook Messenger) is Threats are defined as statements that show inclination to hurt someone. This can include physical harm, but also publication of private information, etc.)
Online distribution of sexually obscene imagery
Distribution of sexual images without the consent of those depicted (minors cannot give their consent) Without actual online distribution, a record does not belong to this category. Illegally recording or threats of distribution are not included. Distribution by the depicted person is not included in this category, unless the depicted is a minor, in which case it is seen as distributed without consent. Webcam sex can be included in this category, if it was between a minor and an adult, or if one of the participants was forced.
Computer trespassing
Logging into an account or computer without the permission of the owner of the computer or account. Suspect changed or removed something from a computer that he/she had no legitimate access to. This can be either via an internet connection or not. If a suspect had legitimate access to the computer or system, but made changes that were undesirable for the victim (for example in case of a jilted system administrator), this is not included.

Table 2.1: Criteria for annotating the data

positive data in the training set.

Since the preliminary results[5] indicated that the number of positive records in some classes was too low to train a well-performing classification model, extra data was annotated for the second phase. A new set of 1200 records was annotated by one of the annotators of the first set, for a total training set of 3096 instances. This new training set was selected in a slightly different manner: the best general online crime model of the first phase (a Random forest classifier with AdaBoost boosting and no resampling, filtered documents and basic tf-idf feature vectors) was trained and tested with 10-fold crossvalidation. In each of the ten testing phases, the instances were separated on whether they were classified correctly or incorrectly. This produced sets of correctly and incorrectly classified instances. For these two sets, an adaptation of the filtering method described in section 3.3 was used to find a list of words that were characteristic for both groups. To select the new set for annotation, the same methods as described for the first set were used, with the additional requirements that the document had to contain at least one of the characteristic words for the classification.

The idea of this sampling method was to annotate more data that either very clearly belonged to a category, or was similar to document that were difficult to classify, and thereby decrease noise and improve classification results. While this method may have introduced some bias, since the data was not randomly selected anymore, the bias is likely to be small: the percentage of documents with interesting labels only decreased using this method.

[5]Discussed in section 4.3.1

	Online threat	**Online distribution of obscene imagery**	**Computer trespass**	**All online crime**
Full set				
Number of records	198	68	80	320
Percentage of total	6.40%	2.20%	2.58%	10.34%
Preliminary data				
Number of records	141	43	64	214
Percentage of total	6.90%	2.27%	3.38%	11.29%

Table 2.2: Categories within annotated data set online crime

Table 2.2 shows the numbers of documents that fall in each of the categories for both phases of the project. Out of the 3096 annotated records, 198 are about online threats, 68 about online distribution of sexually obscene imagery, and 80 about computer trespass. In total 320 documents fall within one or more of these three classes. For the preliminary data set, the percentages are a little higher than for the full set. However, the difference is small, and the increased number of positive instances can still have a positive effect. This is especially the case for the smallest class, for which the absolute count of positive instances went from 43 to 68 documents. This increase gives the model over 50% more positive examples of what might be relevant information to determine how to classify a document.

Figure 2.1 (created using T-SNE) gives an idea of how the full data set is distributed over the feature space. It clearly shows that the data points of the different classes are not each assigned their own part of the feature space. Instead all data seems to be scattered over the space, without significant clustering for each category. This distribution means that there is quite some overlap in the feature values for the different classes, which can be explained by the fact that it is text data. Most features will be insignificant for the classification: for example, the word "you" is common enough that it is unlikely to be relevant for the classification, but it occurs in many documents and thus influences the way the data is distributed in the feature space.

The goal is to create a model that correctly classifies data for belonging to one or more categories of online crime. This can be done through the use of multiple binary models that each determine whether a document does or does not belong to a certain class. This combination of multiple binary classifier is sometimes called a "one-vs-all approach". However, most of the annotated data belonged to none of the categories, which causes the training data to be imbalanced.

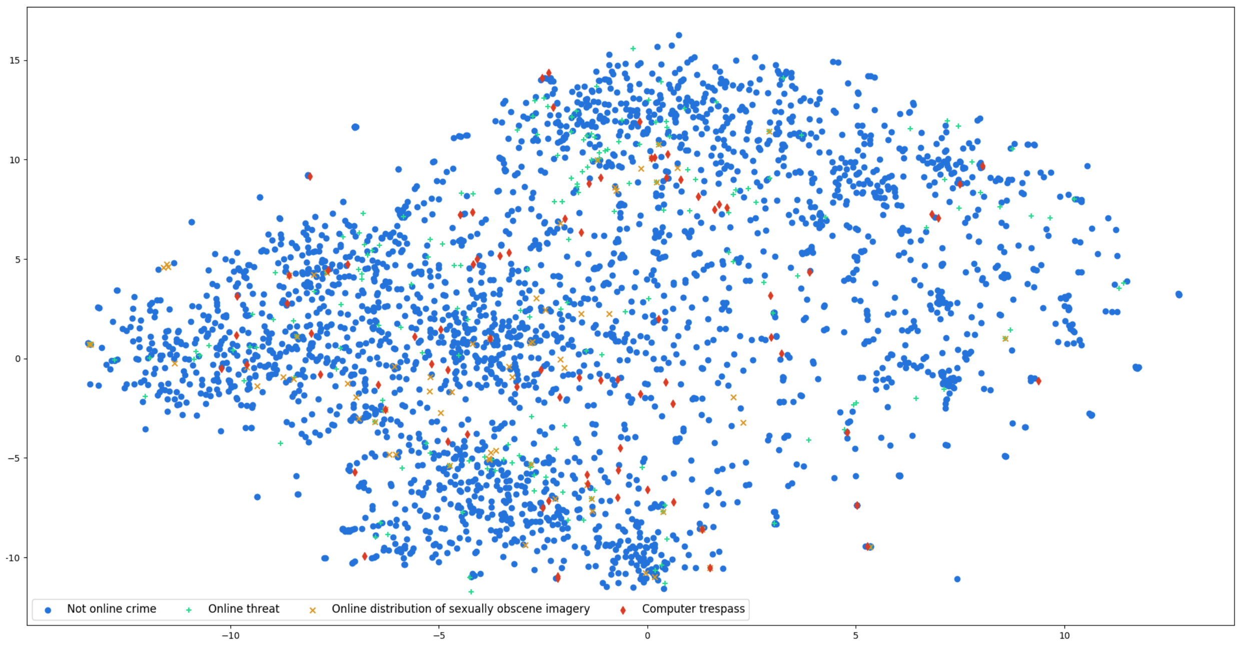

Figure 2.1: 2-dimensional feature space of lemmatized annotated data, created using T-SNE

2.2 Class Imbalance

As shown in table 2.2, the percentage of positive or interesting[6] records is low: just over 10% of the data falls in any category of online crime, and for two of the classes (Online distribution of sexually obscene imagery and Computer trespass) the percentage of positively labeled data is just over 2%. The majority of the records belongs to neither of the interesting categories. This means there is a class imbalance: the data is not evenly distributed over the possible classes. This imbalance can cause the negative samples (those records that do not belong to a category) to have a large influence on the final model. For example, a model trained with imbalanced data can classify everything as negative, since that results in high numbers of correctly classified documents and thus a high accuracy score. Classifying every instance as negative is not very informative, which makes this type of model practically useless.

Another issue that contributes to our class imbalance problem is the size of the training set. Since the training set is not very large to begin with, the class imbalance might make it difficult for the model to discern what features may be relevant for determining whether a records belong to a specific category. Class imbalance is a frequently occurring problem in data mining, and there are some ways to deal with it. Resampling algorithms such as SMOTE [Luengo et al., 2011, Batista et al., 2004] can remedy this problem by artificially creating extra data for minority classes, or removing data points from the majority class. Resampling will be discussed more in-depth in section 3.4. Using different evaluation measures of the model's performance can also help when training a model on imbalanced data. This is discussed further in section 4.2

[6]Documents that belong to one of the categories

Chapter 3

Methods

3.1 Preprocessing

To train a model on text data, some preprocessing is needed to make the data suitable for training the model. Machine learning algorithms generally take numeric data, so the texts have to be transformed into feature vectors. The first preprocessing steps are mostly on a linguistic level, and can improve the classifier by removing unnecessary information or otherwise structuring the data. Most preprocessing is optional and task- or algorithm dependent, so below multiple options are discussed, all of which have been used or considered for this thesis. These include tokenization, lemmatization and PoS tagging. In this project, the tool FROG (developed at Radboud University by Van den Bosch et al. [2007]) was used for most of these preprocessing steps.

3.1.1 Tokenization

One of the first steps of preprocessing is tokenization. Tokenization is the process of breaking up a document in specific part, often words, characters or sentences. While this may seem like an easy task of just chopping the text into pieces at white spaces and punctuation marks, it can be very difficult. In most languages word or sentence boundaries are not defined clearly by white space and punctuation. In English and Dutch punctuation can occur at the end of a sentence or in a contraction of words ('it's', 'Mr.', ''s ochtends'). In other languages, such as Chinese, there is no white space between words within a sentence. Sentences then need to be parsed before it can be broken up into separate words. These problems show the need for tokenization to make the data as uniformly formatted as possible.

FROG uses a rule- and heuristic-based tokenizer, called Ucto [Van den Bosch et al., 2007, van Gompel et al., 2012]. Ucto first splits a text document into fragments based on spaces. These fragments are then compared to language-specific regular expressions. If a match is found, the matching part of the fragment is marked as a word token, and labeled with the matching regular expression. Remaining parts are seen as a separate fragments that are subsequently matched. An example of Ucto's tokenization of the sentence *Mr. Jones's daughter had been sleeping on the sofa* is given in table 3.1[1].

[1]While examples are all in English with the audience of this thesis in mind, it can be assumed that any cases in Dutch would be similar if no examples for Dutch are given as well.

Word	Word-level type	Sentence-level type
Mr.	ABBREVIATION-KNOWN	BEGINOFSENTENCE NEWPARAGRAPH
Jones	WORD	NOSPACE
's	SUFFIX	
daughter	WORD	
had	WORD	
been	WORD	
sleeping	WORD	
on	WORD	
the	WORD	
sofa	WORD	NOSPACE
.	PUNCTUATION	ENDOFSENTENCE

Table 3.1: Ucto tokenization example of *Mr. Jones's daughter had been sleeping on the sofa*

3.1.2 Irrelevant terms

After tokenization it can be worthwhile to remove words that are unlikely to be interesting from the documents. There are multiple ways to do this. The first is to remove stop words. Stop words removal consists of the removal of extremely common words. These words, such as "you", "be" or "it" in English, and "jij" en "hebben" in Dutch, occur so often in the language that they can be assumed to carry no information on what the topic of the text is. Stop words can be derived from the data to get a corpus-specific list of stop words, or a standardized list of stop words can be used.

Part-of-Speech (PoS) tagging can also be used to remove words that have low informative value. A PoS-algorithm determines the part-of-speech tag of each word or word cluster. Since certain parts-of-speech can provide more or less information about the topic of a text (nouns and verbs are usually rich in information on the topic, prepositions usually are not), PoS tagging can be used to filter out words that are unlikely to be useful in text classification, like articles or names [Bonatti et al., 2016]. It also plays a crucial role in the next step of the linguistic preprocessing: lemmatizing the text.

3.1.3 Lemmatization

In text mining the amount of times a word occurs in a document is crucial for building the feature vectors used to train a model. A word that occurs many times in a single document is likely to say something about the topic of the text. However, counting word occurrences in a text document is not as straightforward as it sounds in many cases. In many languages words can be inflected - marked to show tense, grammatical case, or other information - which changes the word from its basic form or lemma. An example for English would be "walk - walked - walking". Since words that are have been inflected usually have a different form than their lemmas, two words with the same lemma are sometimes counted as occurrences of different words. The word frequencies are intuitively more accurate when 'walked' and 'walking' both count toward the frequency of 'walk'. Lemmatization is the process that removes inflection and reduces words to their dictionary form, which can improve statistical counting [Bonatti et al., 2016]. For example, the lemma of 'better' is 'good', the lemma of 'found' is 'find'. This means that after lemmatization, the frequency of 'good' in the sentence *I thought the cake was good, but I liked the ice cream better* would be 2, instead of the 1 of the unlemmatized sentence, since both 'good' and 'better' are counted towards the term

Word form	**PoS**	**Delete**	**Insert**	**Resulting lemma**
bijen	N(soort,mv,basis)	-en		bij
haarspleten	N(soort,mv,basis)	-ten	et	haarspleet
spleten	N(soort,mv,basis)	-ten	-et	spleet
	WW(pv,verl,mv)	-eten	-ijten	splijten
haarstukje	N(soort,ev,dim,onz,stan)			haarstukje
staken	N(soort,mv,basis)	-ken	-ak	staak
	WW(inf,nom,zonder,zonder-n)			staken
	WW(inf,prenom,zonder)			staken
	WW(inf,vrij,zonder)			staken
	WW(pv,tgw,mv)			staken
	WW(pv,verl,mv)	-aken	-eken	steken

Table 3.2: Some examples of rewriting rules for Dutch for the FROG lemmatizer[2]

frequency of 'good'.

Lemmatization is similar to stemming (a technique which removes the inflection from a word), but unlike stemming it bases the lemmas on the context. Ambiguous words are usually lemmatized to the correct lemma, while stemming these words can lead to the wrong words being counted. The word "meeting" can be both a noun and a (form of a) verb. A stemmer reduces both to the same form ("meet"), while a lemmatizer will select the appropriate base form based on context ("meet"(verb) or "meeting"(noun)).

FROG's lemmatizer uses combinations of words and PoS tags to determine the correct rewriting rule for each word [Van den Bosch et al., 2007]. Only the last 20 characters of a word are taken into consideration, as in Dutch all morphological changes are made in the suffix of the word. If there is only a single possible rewriting rule for a word form, that rule is used. If there are multiple options, the rules for the PoS that matches the PoS of the word are selected. Since PoS includes things like tense, this takes care of most options. In those very rare cases where this still leads to multiple possibilities a random option is selected from the remaining rules. Some examples of rewriting rules that FROG uses to lemmatize words are shown in table 3.2. In the (Dutch) examples shown there, we can see that some word forms have a single rule, such as *bijen* and *haarstukje*, and some have multiple rewriting rules for every possible PoS tag, such as *spleten* and *staken*.

3.2 Feature vectors

To use text documents to train a classifier, the data has to be represented numerically. This is done by determining features or properties of the document that can have a certain value. Usually in text mining these features are directly derived from the words in the document, such as the number of occurrences of a certain word in the document. Each document can be represented as a high-dimensional vector of feature values. In this section I will discuss methods to go from text document to ready-to-use feature vector. The mock documents in Example 1 will be used as a running example throughout this section.

[2]Source: FROG Documentation (available at `https://languagemachines.github.io/frog/`)

Example 1

Document 1
The dogs are playing in the garden.

Document 2
The cats were sleeping.

Document 3
Dogs and cats are playing together

3.2.1 Word frequencies

The simplest characteristics of a text document are the words (or letters) contained within it. To create numerical features from these words, word frequencies or word counts can be computed by counting the number of occurrences of a word in the document. Combining the values for different word features in a vector gives us a feature vector that characterizes the document. Each word in the lexicon has its own place on the feature vector, which means the vector will be high-dimensional, since there are often thousands of words in a lexicon. This resulting feature vector is an example of sparse data: most words from the lexicon will not appear in a specific document, and even if they appear, usually not multiple times. As discussed in section 3.1.3, lemmatization can increase the frequencies and reduce the sparsity by mapping all forms of a word to a single lexicon item, while removing stop words can slightly reduce the number of dimensions of the feature space.

An example of the word frequencies for the documents in example Example 1 can be found in table 3.3. The frequencies shown in the table make up the feature vectors: the feature vector for documents 1, 2 and 3 would be

$$\text{document } 1 : < 1, 0, 1, 1, 1, 0, 0, 2, 0, 0 >$$
$$\text{document } 2 : < 0, 1, 0, 0, 0, 0, 1, 1, 0, 1 >$$
$$\text{document } 3 : < 1, 1, 1, 0, 0, 1, 0, 0, 1, 0 >$$

The order of the features in the vector is arbitrary. So while in the above example the features are sorted alphabetically based on the words they represent, this is not necessarily the case. However, the order of the features is fixed across the corpus. If the feature vector for document 1 is based on alphabetically-ordered features, then the same holds for the feature vectors for the other documents.

An alternative is to compute binary counts for the feature vectors instead of actual word frequencies. In binary counts it does not matter how many times a term occurs within the document; instead the only factor that is relevant is whether a term or N-gram does or does not occur in the document. If it does, the value is set to 1, otherwise it is set to 0. A binary feature vector consists of only ones and zeroes and shows what words appear in a document. Using binary counts removes the extra influence of longer documents: longer documents have relatively many re-occurring words, and thus often higher word frequencies. When binary counts are used, it becomes easier to compare documents of different sizes.

Word in lexicon	Doc 1	Doc 2	Doc 3
are	1	0	1
cats	0	1	1
dogs	1	0	1
garden	1	0	0
in	1	0	0
playing	0	0	1
sleeping	0	1	0
the	2	1	0
together	0	0	1
were	0	1	0

Table 3.3: Word frequencies for corpus of example Example 1

N-grams

While word frequencies are crucial to the text mining process, the bare number of occurrences provides very little information about the order and context of the words in the document. N-grams can be used to get a little more information. N-grams are sequences of N consecutive words from a text. To illustrate, the bigrams (2 consecutive words) in document 1 of the example are

the dogs, dogs are, are playing, playing in, in the, the garden

The trigrams (3 consecutive words) for the same document are

the dogs are, dogs are playing, are playing in, playing in the, in the garden

Although the use of N-grams causes the data to be even more sparse[3], it does provide more context in which words are used. For this research, for example, knowing that the bigram “online threats” appears within a document could be considered as more informative than the knowledge of the separate words “online” and “threats” appearing in the document. This is why in some cases, N-grams can improve classifier performance [Joulin et al., 2016]. N-grams frequencies can be computed in the same way as word frequencies, and the two can co-occur. Using word and N-gram frequencies together provides detailed information about words and their direct context. However, there is a downside to the use of N-grams: because there are so many possible combinations of words, the number of features gets very large when N-grams are used, especially for a high N. This can become a problem in a later phase when training the model can demand more RAM than may be available.

3.2.2 Tf-idf term weighting

While word frequencies are a useful way to represent documents, they do not provide any information about the usage of a word in the full corpus. For text mining purposes it is often useful to know whether a particular word is a common word in general or used relatively much in a specific document. Word frequencies do not provide information on this. Applying term weighting can help to represent these types of information, for example by giving commonly used words in the corpus a lower weight. An often-used term weighting mechanism in text mining is tf-idf, term frequency - inverse document frequency [Feldman and Sanger, 2007]. Term frequency, inverse document frequency and tf-idf are defined as follows:

[3]Most combinations of words do not occur multiple times, if any, in a single document

$$
\begin{aligned}
\text{term frequency (tf)} &= \text{number of occurrences of term } t \text{ in document } d \\
\text{inverse document frequency (idf)} &= \log \frac{\text{number of documents in full corpus } D}{\text{number of documents in } D \text{ that contain term } t} \\
\text{tf-idf} &= \text{tf} \cdot \text{idf}
\end{aligned}
$$

The tf-idf value for a word in a document represents the frequency of the word in that document relative how frequent it is across the whole corpus. A high tf-idf value shows that the word appear in a document relatively frequent, and thus can be considered characteristic for the document [Feldman and Sanger, 2007]. A low value means the word appears often in the corpus, or maybe only once in the specific document. If a word does not appear in the document at all, the tf-idf is undefined. This can also be seen as a very low tf-idf score, and is often considered to be 0 for further computations.

An example of some of the tf-idf values for the three-document corpus of example Example 1 are shown in table 3.4. Take the term 'the': it occurs two times in document 1, and it occurs in document 2 as well. The term frequency for 'the' in document 1 is 2, and the inverse document frequency is $log\frac{3}{2} = 0.176$. This means the tf-idf value is $2 \cdot 0.176 = 0.352$. All other values can be computed in the same manner. The tf-idf feature vector consists of all these values in a set order, similar to the word frequency vectors.

Word in lexicon	**Doc 1**	**Doc 2**	**Doc 3**
are	0.176	-	0.176
cats	-	0.176	0.176
dogs	0.176	-	0.176
garden	0,477	-	-
in	0,477	-	-
playing	-	-	0,477
sleeping	-	0,477	-
the	0.352	0.176	-
together	-	-	0,477
were	-	0,477	-
are playing	0.176	-	0.176
cats are	-	-	0,477
the dogs	0,477	-	-
⋮	⋮	⋮	⋮

Table 3.4: Tf-idf values for some words and N-grams from example Example 1

Using tf-idf to build feature vectors from documents has an important additional benefit: very common terms will automatically be less important, as those will occur in almost every document and therefore will have a very low tf-idf value. In the same way words that are rare but occur often in a single document are 'boosted' with a high tf-idf value. This means that it is possible to get a feeling for what the most characteristic terms for a document are just by looking at its tf-idf feature vector.

As with word frequency vectors it is possible to take binary counts instead of term frequency when tf-idf is calculated. In that case the *tf* is replaced by a 1 if the word occurs in the document, or a 0 if it does not.

3.3 Feature reduction and selection

Since text document usually have thousands of features, it is often useful and efficient to somewhat reduce the feature space. This can make the process of training computationally easier (since less features mean less possibly relevant characteristics to take into consideration), and helps prevent overfitting the model because the options for the model are more limited. In document classification, often many words can be dropped without harming classifier performance [Feldman and Sanger, 2007]. The stop word removal mentioned above is a very simple way to remove some features. However, stop word removal usually discards only a small number of features. Sometimes it is possible to remove more features by computing their relevance and keeping only the most relevant features.

There are three types of feature selection approaches: filter, wrapper, and hybrid [Liu and Yu, 2005]. Filter approaches make a selection based on data characteristics. Wrapper approaches use some mining algorithm to evaluate relevance, instead of the immediate properties of the data, but can be computationally inefficient. Hybrids combine the two types.

A method to reduce the number of features that is used in this research is based on tf-idf values. As a first step, the data for a binary classification task is split into two subsets: the 'positive' and the 'negative' data[4], in the form of the tf-idf vectors for all documents. For each subset, the mean feature vector is computed: this vector contains the mean value for each feature over the whole subset. This results in two feature vectors: one representing the average positive document, and one representing the average negative document. The values of these two vectors can then be compared. If there is a large difference between the mean tf-idf values of a term in the two subsets, this may indicate that the term is characteristic for either the positive or negative set. By taking as features only the frequencies of these terms, the feature space could become more clearly divided into separate areas. For this thesis, the filtering factor was set to 3. This means that all terms of which the positive and negative mean tf-idf values have a difference of factor 3 are kept, and all other terms are discarded. So, with P_t and N_t as the positive and negative average tf-idf values for term t:

$$t \text{ is } \begin{cases} \text{kept} & \text{if } P_t \geq N_t \text{ or } N_t \geq P_t \\ \text{discarded} & \text{otherwise} \end{cases}$$

Applying this filter to the text documents usually discards the majority of the terms, since most terms in a corpus do not appear significantly more often in on type of documents than in another. In a way, this method is equal to generating a corpus-specific list of stop words and filtering out those words. The main difference is the restrictiveness: applying this method on the police data set left us only 2,878 out of 41,548 features for the general crime category. For online threat, online distribution of sexually obscene imagery and computer trespass the remaining lexicon sizes were 2,425, 1,594 and 1,854 terms, respectively. This extreme reduction of the amount of features can have a very large influence on the final results. This is amplified by the fact that only the most relevant terms remain.

[4]Positive here means that a document belongs to the online crime category (the documents that are interesting), while negative means it does not

3.4 Resampling

As described in section 2.2, the data set in this project has a large class imbalance. Since most machine learning algorithms were developed with an equal distribution over the possible classes in mind, this can make it difficult to create a model with optimal performance when data is imbalanced. An optional step to help remedy this problem is to resample the data. Resampling is used to get a more balanced set of training data by increasing the number of samples in the minority class, or decreasing samples from the majority class. These two approaches can be distinguished in the two main types of resampling algorithms: oversampling and undersampling algorithms.

3.4.1 Oversampling

Oversampling is used to to get more instances of data of a minority class. This can be done by simply performing sampling with replacement of minority class instances, which is sometimes called "bootstrapping". This method is fast and efficient, but it comes with a large drawback: because some of the samples in the minority class are sampled thus observed multiple times in the training phase of the model, while others might not be observed even once, there might be a higher chance of overfitting [Batista et al., 2004]. However, this simple oversampling method usually performs quite well on data sets with minority classes that are not too small.

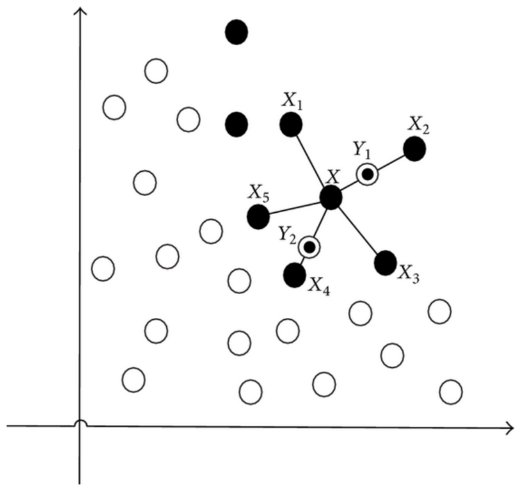

Figure 3.1: Illustration of the SMOTE oversampling algorithm [5]

Another oversampling method is to create synthetic samples based on the samples in the minority class. SMOTE (Synthetic Minority Oversampling TEchnique) is such an oversampling algorithm. SMOTE first creates a high-dimensional feature space based on the features of the data set. All instances in the minority class are mapped onto this feature space, which can then be used to create a new sample. The first step of creating a new data point is to randomly select one of the existing samples, X. From the k nearest neighbours of X in the feature space, a second sample, X_i, is selected randomly. By then choosing a random point on the line segment between X and X_i, a new data point Y can be generated. The exact feature values are determined by the exact position in the feature space. This process is repeated for different existing samples X, until the needed amount of

[5]Image from Xie et al. [2015]

synthetic samples is reached. An illustration of the problem is given in figure 3.1. The idea behind this method is that points between two instances of the same class will also be of that class. Batista et al. [2004] have shown that SMOTE performs significantly better than bootstrapping, mainly since it is less prone to overfitting the training data. However, it is computationally expensive for data with many features, since all features are taken into account when the new instances are created.

3.4.2 Undersampling

Undersampling methods remove data from the majority class from the training data set. This removal can be random deletion of majority class samples, to try to get more balanced data. However, this is risky, since it might lead to discarding very characteristic samples from a class. Some type of heuristics can be used to try and remove only uninteresting examples, outliers or noise. The downside of using undersampling algorithms is that information is removed from the training set. Especially if the training set is not very large to begin with, it is usually advised to be careful about using undersampling to get a better ratio between classes, since it may cause the data set to become too small.

An undersampling algorithm that aims to decrease noise in data is ENN. ENN uses Wilson's Edited Nearest Neighbour Rule [Wilson, 1972] to determine whether samples are typical for the class they belong to. Similar to SMOTE, the first step of the ENN algorithm is to create a feature space. The samples of all classes are mapped onto this feature space. For a randomly selected sample X, the k nearest neighbours are determined. The class labels of these samples are compared to the class label of X. If the majority of the neighbours' classes is different from the class of X, X is removed from the training data [Batista et al., 2004]. This is repeated until the required amount of samples is reached. The underlying idea is that this process removes samples that have comparable feature values to instances from another class, thereby removing noisy, ambiguous data. ENN can be applied to remove samples from any class that differ in class label from their neighbours. In that case, it is more of a data-cleaning than an undersampling algorithm. However, it is also possible to only remove samples from certain classes (usually the majority class). This helps ensure a more balanced set of training data.

It is possible to combine several types of resampling algorithms. In this project, SMOTE-ENN was used, which first uses SMOTE to create synthetic minority class samples, and then removes some majority class samples using ENN. This method was described by Batista et al. [2004], who showed that the combination of SMOTE and ENN usually performs well for very small minority classes (100 positive instances or less).

3.5 Machine Learning

After feature selection, a classifier model can be trained with on a machine learning algorithm. This algorithm takes the training data, and uses it to train a model which can be used for the classification of new data. The goal is to create a model that predicts the correct label for each feature vector, even the ones that do not appear in the training data. In this section I discuss some of the most popular algorithms for text classification. Most of these algorithms are supervised learning algorithms, in which all training data is labeled before the start of the training phase. Since this project had so much unlabeled data available for training as well as labeled data, some semi-supervised learning algorithms will be discussed in section 3.5.3. These methods use both labeled and unlabeled data to train the classification model.

3.5.1 Supervised learning

Supervised learning algorithms use only labeled training data to train the model. The input for these types of algorithms is a set of feature vector-label pairs. Although it is possible to train models that are able to classify multiple classes at once using multiclass classification algorithms, this thesis has selected a very simple approach: the multiclass classification problem is split into three independent binary classification problems, as described by Hsu and Lin [2002]. These three problems are whether a document belongs to the either of the three categories mentioned before. This means there is one model to determine whether a document is about online threats or not, then a second and third model to determine whether the same document belongs to the categories of online distribution of sexually obscene imagery and computer trespass, respectively. Therefore this section will only discuss algorithms for binary classification[6], including Naive Bayes, Random Forest and Support Vector Machine classifiers.

Naive Bayes

Naive Bayes classifiers are a type of statistical probabilistic classifiers. These classifiers calculate class probabilities based on the feature values of the training data and the vector that is being classified. Naive Bayes classifiers combine Bayes' theorem[7] with the assumption that all features of an object are mutually independent. By assuming the mutual independence of the features, Bayes' theorem can be used to compute the class probabilities from the feature values. Without this assumption, the computation would become very complex, since it would require the dependencies between the features. These dependencies are difficult to estimate, and with large amounts of features this would be computationally expensive.

The assumption of mutual independence is usually very unrealistic in text mining, since linguistic expression are built in a certain way which causes the separate words to be related and not independent. However, even though the assumption is false, Naive Bayes classifiers are relatively efficient and effective due to their simplicity, and therefore are often used in text categorization tasks [Nigam et al., 2000, Tang et al., 2016]. Their performance usually is best with large data sets, but it is possible to get good results even when the training set is smaller.

The formal definition of Naive Bayes is as seen here, where $x_1, \cdots, x_n$ are the features (in text mining these are usually term frequencies or tf-idf values), and C is the class that is being considered:

$$P(C|x_1, \cdots, x_n) \propto P(C) \prod_{i=1}^{n} P(x_i|C)$$

Calculated here is the probability of the document being of a certain class ($P(C|x_1, \cdots, x_n)$), based on the probabilities of the separate words of the documents for that class. The product of the probabilities of all separate words given a class ($\prod_{i=1}^{n} P(x_i|C)$)[8] is multiplied with the probability of that class ($P(C)$). For every document this probability is calculated for each possible class, after which the class with the highest probability is used to classify the document.

[6]Some of these algorithms are adaptable to multiclass classification problems with minor alterations

[7]$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$

[8]Based on relative word count of documents in the training set

There are multiple variations of Naive Bayes models, each with their own strengths. These differ mainly in the calculation used to compute feature probabilities. Examples include Bernoulli Naive Bayes, Multinomial Naive Bayes and Gaussian Naive Bayes. It has been shown that Multinomial Naive Bayes classifiers usually perform better than other variations for sparse data sets such as text data [Tang et al., 2016]. These classifiers assume that term frequency within a document satisfies a multinomial distribution. The probability of a term given a class in a Multinomial Naive Bayes model is defined as follows by Manning et al. [2008], with term x, class C, $|X|$ as the vocabulary size of C and $|F_{xC}|$ as the number of occurrences of x in C.

$$P(x|C) = \frac{|F_{xC}|}{\sum_{i=1}^{|X|} |F_{iC}|}$$

This means that the probability of a term given a class in Multinomial Naive Bayes is the number of occurrences of that term in the class divided by the total number of occurrences of terms in that class[9]. It is easy to see that this method for calculating probabilities is intuitive for text classification: if a term occurs often in a document, the chance of it being relevant for the topic of the text is higher as well.

Naive Bayes models can be combined with smoothing. Smoothing adds some constant value or percentage to each word frequency, to prevent words that do not appear in the training set from introducing the value zero into the product of probabilities. This would cause documents with terms that did not occur in the training data to not be classified properly, since the probabilities of these words are set to zero for each class. Smoothing solves this problem by applying some extra value to a probability, making sure that the probability of an occurrence can never be zero, and thus making sure that classification can happen.

Support Vector Machines

Support vector machines (SVMs) were first described by Cortes and Vapnik [1995] as a method for non-probabilistic binary classification. To build SVMs first a high-dimensional feature space is created. The feature vectors from the training set are mapped onto this feature space using a non-linear mapping or kernel function. In this feature space a optimal separating hyperplane can be constructed to separate the points belonging to two classes (see figure 3.2). This optimal hyperplane maximizes the distance to the training data of both classes, so it is placed in such a way that each data point is as far as possible from the hyperplane separating the two areas. The goal of this maximization is to minimize noise.
It is always possible to separate two classes by a hyperplane when using a nonlinear mapping in a feature space with sufficiently high dimensionality [Han et al., 2011]. This means dimensions can be added to separate the instances of the two classes. An example of this can be seen in figure 3.2, where a 2-dimensional space is transformed to a 3-dimensional space to accommodate the separating hyperplane. According to Han et al. [2011], the specific kernel function chosen for the mapping process usually has limited impact on the resulting model's accuracy.
Classification of a new data point is done by mapping it onto the same feature space, using the same non-linear mapping. Based on its location in the feature space respective to the hyperplane the new feature vector can be classified in one of the two classes. An important advantage of SVMs is that they function well in large feature spaces, which makes feature selection less essential [Hotho et al., 2005]. Since text data often had very large numbers of features, this makes SVMs

[9]Which is the sum of the term count of all documents in class C.

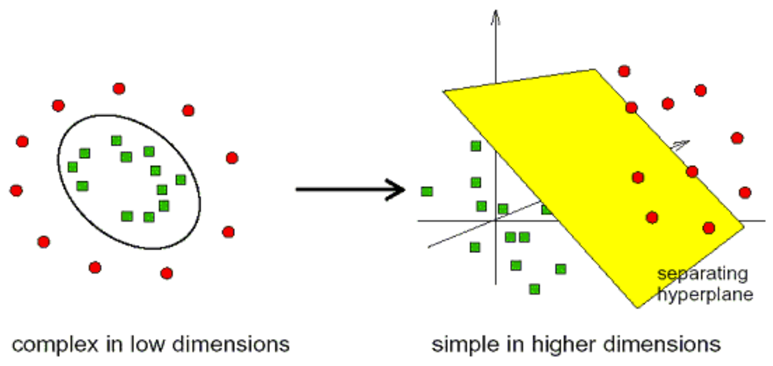

Figure 3.2: Example of high-dimensional feature space and hyperplane for support vector machine[10]

very interesting for text classification tasks. The algorithm also has its downsides: SVM training is very memory-intensive, and it can be difficult to determine what kernel to use for the mapping process.

3.5.2 Ensembles and boosting

There are some methods to increase the performance of a classification model. One of these methods is ensembling. An ensemble is a composite model, built from a combination of multiple less complex classifiers [Han et al., 2011]. The models that form an ensemble model are often trained on slightly different parts of the data set. How the selection is made can vary greatly depending on the algorithm used. Some examples for the training set selection for Random Forests, AdaBoost and XGBoost are given in the rest of this section. The output from the different simple classifiers is combined to get the output of the ensemble classifier. The simplest method to combine the outputs is majority vote, but weighted voting is also common. The intuition behind this is that a group knows more than an individual: as long as the majority of the base classifiers gives the correct classification, the total classification will be correct as well. Ensembling also prevents overfitting: since each base classifier sees only part of the data, the chances of the model becoming too much tailored on the training data are slim.

An ensemble is often more accurate than the base classifiers that it consists of, if these base classifiers meet certain requirements. The first requirement is that the base classifiers should perform better than random classification. If many of the base classifiers do not perform better than random, the chances of their combined input (for example, in a majority vote) being correct are low. Second, the base classifiers should have relatively little correlation between themselves. No two should be exactly the same, since that would lead to a bias. This is why the different base classifiers are trained on different parts of the data in many ensemble algorithms.

A special category of ensemble algorithms is boosting. There are many different boosting algorithms that differ in their exact approaches. The main intuition of boosting is similar to that of general ensembling. However, boosting accomplishes increase of performance by training multiple models on slightly different data, and take the *weighted* combination of their classification results as the final output. In this section, two boosting algorithms will be discussed: AdaBoost and XGBoost.

[10]Source of image: `https://inovancetech.com/how-to-trade-rsi.html`

Random Forests

Random Forest classifiers are a combination of multiple (much more simple) decision tree classifiers. Decision trees are tree-structured models. Each split of of the tree divides the data set into multiple subsets based on the value or presence of a certain feature. At the ends, the leaves of the tree, the subsets consist (mostly) of data from a single class. Classification of a new instance is performed by going through the tree, and at each split taking the path corresponding to the feature values of the instance. The instance is then classified as the majority class of the leaf where it ends up. A (bogus) example of a very basic decision tree is given in figure 3.3. When starting at the arrow, the presence of several words is checked with the features of the new instance. The path through the tree for an instance depends on the feature values. In the final leaf, the instance gets one of three labels: "not fast food", "burger" or "pizza".

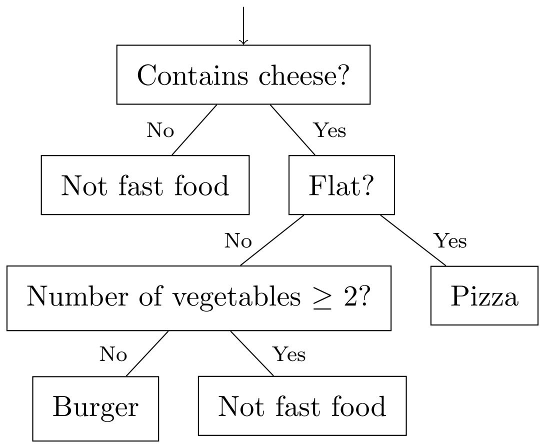

Figure 3.3: Example decision tree for classification of fast food articles (not based on actual results or data)

While decision tree classification is very intuitive and efficient, it is prone to overfitting the training data. This means that the tree has become too specialized for the training data, which can cause a loss of generalization power in the model. For example, in the training data fast food might not contains vegetables, while there are fast food articles that do, causing all these items to be wrongly classified. Another downside of decision tree classification is that the models often have too low complexity to adequately perform more complex classification task such as text classification. Random forests were designed to solve these issues [Breiman, 2001].

Random forest classifiers function by training multiple decision trees. Each tree is trained on only a subset of the total set of features. Additionally, only a random sample of the training data is used to train each tree, instead of the full training set [Breiman, 2001]. New data points are classified by performing the classification with every decision tree, and then putting those classes to a majority vote: the data point is classified as belonging to the class most individual decision trees labeled it with. Because each tree in a random forest classifier only uses a subset of the features and a subset of the training data for training, they are unlikely to overfit. By using only part of the training data, it becomes very difficult for the model to be too specialized. This increases the accuracy of the random forest classifier on previously unseen data and makes it less susceptible to overfitting. This makes Random Forest classifiers a popular choice. It does have some disadvantages: training the models requires much memory and computation power, since all trees are trained in parallel.

AdaBoost

AdaBoost, short for "Adaptive Boosting" [Freund and Schapire, 1997], is a meta-algorithm to improve classification results. It trains multiple classifiers, each on a sample of the full training set. The AdaBoost algorithm starts by assigning weights to every instance in the full training set. Initially, all instances get equal weights. To train a each model, a sample S_i is taken from the whole set of training data, based on the weight of the individual instances. The higher the weight, the higher the chance of the instance appearing in the sample. For the initial phase, all feature vectors are given equal weight and thus have equal chance of appearance. The sample is used as training data to train a basic classifier[11]. The result is a classifier C_i. This classifier is tested on its own training data S_i. The weights of the instances in the sample are adapted to represent the classification: if an instance was classified correctly, its weight is decreased, if it is classified incorrectly, the weight is increased. In the following steps, new classifiers are trained on a new sample [Schapire, 2013]. This places the focus of the classifier on the difficult cases.

The final classification is made by combining the results of the whole set of classifiers. However, the output from each classifier is weighted by its performance. Classifiers with high accuracy have higher weights, and therefore a bigger influence on the final classification. For classification, this means that the weights for all classifiers that assign a certain class to an instance are summed, and the class with the highest total weight is assigned in the final classification of the AdaBoost classifier [Han et al., 2011].

When the base classifiers used in the ensemble already perform reasonably well, AdaBoost often increases performance. However, the algorithm is susceptible to noise [Wang et al., 2009], since outliers and atypical instances can also be included in the sample, and have a large probability of being assigned higher weights. This causes the model to be overfit on those instances.

XGBoost

XGBoost (eXtreme Gradient Boosting, as described by Chen and Guestrin [2016])is a gradient tree boosting algorithm that was used to win several machine learning challenges. It is much faster than most other resampling algorithms and able to perform well for sparse data sets, which makes it a good fit for text data. XGBoost has similarities to both Random Forest and AdaBoost classifiers, since it trains multiple tree classifiers successively and combines their output to get the final classification. The XGBoost algorithm uses regression trees. These models are similar in structure to decision trees, with the main difference being the output: decision trees give each input instance a class label as output, while the output of a regression tree is a class probability.

The global idea of training an XGBoost model is as follows. The training data set is used to train a single regression tree[12] C_1. The performance of C_1 is tested using the training set, outputting a value for each data point. For binary classification, this value p_1 is the probability that the data point belongs to the interesting class. It then calculates for each data point how much the predicted probability value p_1 differs from the actual value l_1 (which will be 1 if the instance does and 0 if it does not belong to the interesting class). This means that after the first round of testing it is known how large the error of the first regression tree is for each training instance. These error values $p_1 - l_1$

[11]Any of the previously mentioned algorithms could be combined with AdaBoost

[12]For more background on how the regression tree is built and how the splits are computed, see Chen and Guestrin [2016]. Instead of regression trees, linear classifiers can be used as XGBoost's base estimators as well.

are used as the goal values or labels for the data for training a second classifier(/estimator). Thus, the second estimator tries to predict how much the prediction of the first classifier will be off the mark. This continues for several iterations[13], with each new estimator predicting the error of the previous one. The goal is to minimize the sum of the errors of all estimators together.

Classification of a new instance is done by using all estimators to give a value. These values are weighted by the complexity of their corresponding estimators: complex models get a lower weight, while simpler models get high weights. This prevents the complete XGBoost model from overfitting. The weighted outputs are then summed to get the final probability for the classification.

To clarify this, an example is given in table 3.5. In this example, there are five documents. As the second column shows, documents 1, 2 and 5 are in the interesting class, while documents 3 and 4 are not. The third column then gives the probabilities p_1 that the document is in the interesting class as calculated by the first regression tree of the model. Let's look at document 1. p_1 is 0.8, which means the first regression tree gives it a high probability of belonging to the interesting class. However, the error E_1 is 0.2, since the estimated probability is not equal to the label. The next classifier gets this value E_1, as its training label. After training, this classifiers outputs p_2 0.1, which gives an error E_2 of 0.1. This continues for the third classifier, which predicts a value P_3 of -0.1, which equals an error E_3 of 0.2. The classification of document 1 is then done by summing the probabilities outputted by all regression trees. In this case it would be $0.8 + 0.1 + -0.1 = 0.8$ (assuming that all regression trees have equal weight). So, the document would be classified as interesting.

Document	**Label** l_1	**Predicted probability** p_1	**Error** E_1	**Predicted value** p_2	**Error** E_2	**Predicted value** p_3	**Error** E_3
Doc 1	1	0.8	0.2	0.1	0.1	-0.1	0.2
Doc 2	1	0.75	0.25	0.05	0.2	0.15	0.05
Doc 3	0	0.15	-0.15	-0.2	0.05	0	0.05
Doc 4	0	0.2	-0.2	-0.3	0.1	0.15	-0.05
Doc 5	1	0.9	0.1	0.4	-0.3	-0.15	-0.15

Table 3.5: Example for first three iterations of XGBoost classification (not based on actual data)

3.5.3 Semi-supervised learning

Because the data set consists mainly of a large amount of unlabeled data, using semi-supervised learning methods might be worth including in the experiment. These approaches combine labeled and unlabeled data in the training phase. While using unlabeled data might seem counter-intuitive, it can increase classification accuracy. One way unlabeled data might provide extra information is information about the joint probability distribution. After using the labeled data to extract information about relevant features, the unlabeled set (which is often much more data than the set of labeled items) might provide features that seem correlated to these earlier features.

Expectation Maximization

The expectation maximization (EM) algorithm is a useful method to use unlabeled data to find a local optimum for the model [Nigam et al., 2000]. First an initial model is trained on labeled training data, using any supervised algorithm. Then this first model is used to classify the unlabeled

[13]One can specify how many estimators should be trained, with an early stop if results don't change over several iterations

data (the E-step), after which all data (both the training set and the newly labeled larger data set) is used to train a new model (the M-step). This model classifies the previously unlabeled data again. These steps of classifying the data, then training a new model using the new class labeling, are repeated until the process converges: until the classifier parameter values do no longer change [Feldman and Sanger, 2007]. At this point a local optimum is reached.

The specifics of the algorithm are not discusses here, since there are many different implementations of the EM algorithm, none of which were used for this project. For an implementation of EM with Naive Bayes, readers can turn to Nigam et al. [2000], who compared multiple implementations of EM. They found that weighting the unlabeled data to be less influential than labeled data can further improve the classification results. Since there is usually much more unlabeled data than labeled data, the information of the labeled data can be easily overwhelmed, even though it is most likely more accurate. Giving the unlabeled data a lower weight than the labeled data can solve this problem.

Label propagation

Label propagation algorithms are based on the assumption that two points that are close in the feature space will belong to the same class. So, when we have a large amount of unlabeled data, and some labeled instances, we can label the unlabeled data by looking at the labels of the instances closest to them.
The general process for label propagation starts with constructing a feature space containing the feature vectors of both labeled and unlabeled data. Similarly to the SVM algorithm, a mapping function is used to create this feature space. Using some algorithm (for example the -nearest neighbour algorithm) the feature vectors are linked to those close to them. Another option for connecting the instances is to create a fully connected graph between the feature vectors with edges weighted to represent how much two data points are alike [Zhu and Ghahramani, 2002]. For each data point, probabilities for each possible class are stored. Usually all probabilities are initialized to equal values. The labeled nodes then propagate or push their labels to the connected nodes: each neighbouring node gets increased probability for the label of its neighbour. The exact increase or decrease of the probabilities is heavily dependent on the implementation of the algorithm[14]. The class probabilities for each node are normalized after this push. The labeled data is then clamped, that is, their class probabilities are reset to their original values, to keep them from becoming influenced by their neighbouring data points. This process is repeated until the probabilities converge, after which each node is assigned the class for which its probability is highest.

A simplified example of label propagation is shown in figure 3.4. The distance between the nodes corresponds with the weights of the connections. In figure 3.4a, there are two labeled instances, and six unlabeled instances. After the first propagation step, the labels are pushed through the connections, leaving only three unlabeled nodes. In figure 3.4c, the last instances are labeled, ending the process. The instance on the top right that is labeled dark red in this last step gets this label because of the weights on the connections: both labels are pushed by a neighbour, but the red label is assigned the highest probability, since it was propagated by the closest neighbouring instance.

Label propagation can outperform SVM when there are few labeled data points [Chen et al., 2006]. Although there are obvious reasons why the algorithm will work best with data that is very clearly

[14]One description of a specific label propagation algorithm is given by [Zhu and Ghahramani, 2002].

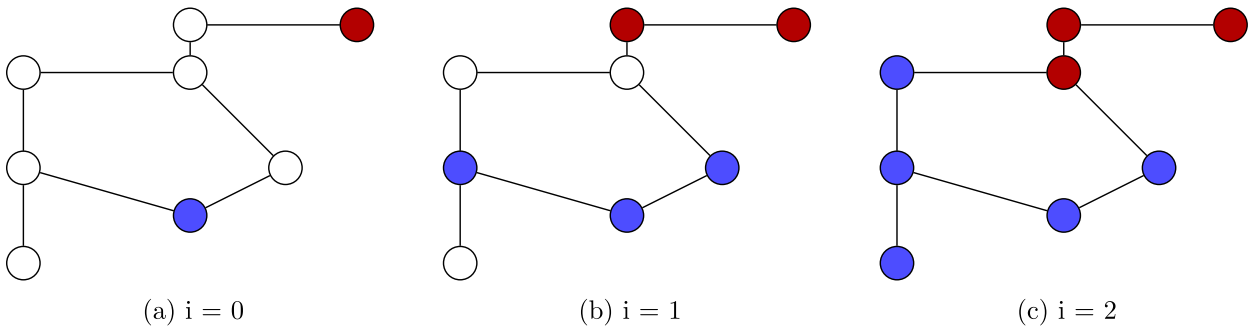
Figure 3.4: Label propagation example. In the first step only 2 instances are labeled, during the second and third step the labels are propagated through the connected neighbours

clustered, it might be interesting to use a multi-label label propagation approach, such as the one described by Kang et al. [2006]. This approach simultaneously propagates multiple labels.

3.5.4 Tested algorithms

Most of the algorithms described in this section were used in the experiment. Each of the algorithms has properties that seem to fit well with the data described in section 2.1. Below a short motivation for each algorithm is given, as well as some background on why other algorithms were not included in the experiments.

As discussed earlier, text data usually contains many features, since the features are based on the occurrences of different words. With this data the ability to handle large feature sets is a big advantage. Random forest classifiers are very efficient on data sets with many features, since only a limited amount of features is taken into account for each tree, and by extension for the final classification model [Han et al., 2011]. With the additional advantage of a low chance of overfitting, these classifiers are an interesting option to examine. The same goes for SVMs: these deal well with large numbers of features, since they use non-linear mapping to find a separating hyperplane [Hotho et al., 2005, Joachims, 1998].

Moreover, text data is usually sparse, since the lexicon is large and the number of different words that occur in a document is not. Naive Bayes classifiers usually perform reasonably well on text and other types of sparse data [Tang et al., 2016, Chen et al., 2009]. This makes them good candidates for the sparse data seen in this data set. In this project mostly multinomial Naive Bayes classifiers were trained, as these perform better for sparse data than Gaussian or multivariate Bernoulli Naive Bayes models [McCallum et al., 1998]. The same argument is important for XGBoost. XGBoost has been used more and more recently, and has performed well for many different tasks, even winning some Kaggle competitions with text mining solutions. Its performance with sparse data is very good. Additionally, it is interesting to compare the results of the three aforementioned models (which are all already established as text mining algorithms) to a high-performing newer algorithm.

AdaBoost was selected to examine the effect of boosting. AdaBoost is a popular boosting algorithm and was chosen because it has been shown to perform well on text classification tasks [Schapire and Singer, 2000]. It can also be combined with different simpler models, which is a way to incorporate the advantages of these models into the ensemble.

There are many other common algorithms that can be used for text classification tasks. The specific data set of this thesis project, however, puts some limitations on which algorithms might perform well. Because the data is sparse, with many features, algorithms that work well on other types of problems might perform very differently for text mining problems. Two examples of algorithms that do not match what is known about the data are neural networks and K-nearest neighbour algorithms.

Neural network approaches require a large set of training data to get good results. Since in this project the amount of annotated data is limited, the chances of a neural network yielding good results are very slim. Additionally, neural networks are difficult to interpret and take a long time to train. Because of these combined factors, these models were not considered in this project.

The K-nearest neighbour algorithm creates a high-dimensional feature space based on the features of the training data. The feature vectors from the training set are then placed within this feature space. When a new, unlabeled vector is presented for classification, the k vectors closest to this vector in the feature space are selected. The classifier predict the class for new data by a majority vote from its neighbours [Hotho et al., 2005, Feldman and Sanger, 2007]. K-nearest neighbour classifiers are simple, but they perform well mostly if the classes each cover a distinct area of the feature space. The data used in this research project contains some noise, since the documents differ majorly in form, length and the words they contain, even within categories. As can also be seen in figure 2.1, it is difficult to distinguish specific areas in the feature space that correspond to a certain class. Instead, the classes seemingly overlap. Because of this, a K-nearest neighbour algorithm seems unlikely to perform well for this data.

Finally, the semi-supervised algorithms mentioned above seemed like a good option, given the data. There was a lot of unlabeled data available alongside with the limited amount of labeled data. However, this proved to be very computationally expensive. Since the amount of memory available was limited to that of a single desktop computer, it was not feasible to train these models within the scope of this project.

Chapter 4

Experiments

4.1 Setup

The goal of the experiment was to create a model to accurately classify police records in belonging to three separate categories: online threats, online distribution of sexually obscene imagery, and computer trespassing. Any records can belong to zero or more of these categories. To explore the possibilities of creating such a model from the available data, multiple combinations of the preprocessing steps and algorithms discussed above were used. Some combinations were left out because of limitations to time or computational power. Because the amount of data available for training was relatively small, four different binary classifiers were trained for each combination: one for each of online threats, online distribution of sexually obscene images, computer trespassing and general online crime (which combines the three other categories). For the first three categories, this choice was mostly about convenience: there are more algorithms for binary classification than for multi-class classification. The fourth category was added as a safeguard in case of too little data: since there are very few positive samples for each of the three main categories, training a classifier for general online crime was a good method to see whether at least the 'online' aspect could be captured by a model.

All parts of the experiment were performed using Python. Python's libraries and options for text processing and data mining made it preferable over other programming languages such as R. The linguistic preprocessing steps, such as tokenization, lemmatization and PoS-tagging, were done using the FROG, a tool for Dutch language processing developed at Radboud University, Nijmegen [Van den Bosch et al., 2007]. In some cases these documents were used as is, in other cases a further selection was made by application of term filtering as discussed in section 3.3. The documents were then transformed to feature vectors based on either word frequencies or tf-idf values by using the *Scikit-learn* FEATURE_EXTRACTION library. The last preprocessing step was done using *Scikit-learn*'s IMBLEARN library and applied one or more of the resampling algorithms described in section 3.4.

To preprocess the text data, the documents were first tokenized and lemmatized using Frog, as discussed in section 3.1. During this phase the interpunction, sentence end markers and names were filtered using Frog's PoS-tagging system: all terms that had no PoS tag or one indication punctuation or names were removed. While this did not work perfectly, it removed most names from the documents. This was desirable since some of the documents in the data set covered the same cases. If names were included, the resulting model might determine some name a relevant feature for classification, causing new documents containing that name to be classified similarly. Since names are not unique identifiers, this was deemed unwanted behaviour and prevented by preemptively removing the names from the text data. In the cases where it was necessary, the filter described in section 3.3 removed all terms that were deemed irrelevant: this were all terms that had an average tf-idf value for one class that was less than 3 times as high as the average tf-idf value for the other class.

Next, the lemmatized documents were transformed into feature vectors. During this process, the words were lowercased, to remove the chance of capitalized, uppercased and lowercased words not being considered as occurrences of the same term. Any features that occurred in less than 0.2 percent of the documents were excluded, to prevent the number of features occurring only once from becoming too high. To determine the differences in performance between binary and non-binary word frequencies, and basic word frequencies and tf-idf values, for each model all four combinations were tested: binary word frequencies, basic word frequencies, tf-idf values based on binary frequencies and basic tf-idf values. After the feature vectors were created, resampling was performed. In the case of SMOTE and bootstrapping the oversampling was performed until a 0.5:1 ratio for the minority/majority classes was reached. Undersampling in ENN stopped when there were no more instances of which all nearest neighbours were of a different class.

Using the created feature vectors, classifiers were trained with four different algorithms: Naive Bayes, Random Forest, Support Vector Machines and XGBoost. Each of these algorithms was also combined with AdaBoost, to examine the effect of boosting on the data. The exception to this were the XGBoost models, which were not combined with AdaBoost to prevent the process from becoming to lengthy by combining two boosting algorithms. The training set was not weighted: all instances initially had equal weight. Similarly, classes were not weighted.
To ensure that scores would not depend too much on the specific distribution of the data set over training and test sets 10-fold crossvalidation was used. For 10-fold cross-validation the data is split in 10 equal parts. Then 10 separate classifiers are trained, each using a different part as its test set, and the remaining 9 parts as its training set. By taking the average score of the 10 classifiers the effect of data distribution is diminished, improving the reliability of the results.

4.1.1 Algorithm parameters and tuning

Most machine learning algorithms have several parameters that can be tuned to obtain the best possible combination for the data used. Since in this project, there were already many larger steps to test, like different algorithms, preprocessing steps and methods of building feature vectors, there were limitations on what combinations could be tested. This led to the following setup: the first step was to determine what type of models performed best with a standard set of hyperparameters. The second was to try different combinations of hyperparameters to tune the best-performing models. In this section the hyperparameters for the different algorithms will be discussed. The values mentioned below are mostly the standard parameters of the *Scikit-learn* libraries. While some tuning was done on the best models, this did not have any significant positive effect on the results, and so the other values will not be discussed here in-depth.

The Naive Bayes models all used Laplace smoothing to prevent feature probabilities of 0. This form of additive smoothing adds 1 to all term frequencies. Uniform prior probabilities were used to initialize the probabilities.

Random Forest have a more complex set of hyperparameters, since the decision trees themselves need to be built according to certain requirements. Each individual tree was assigned no maximal depth or number of leaf nodes, but a split into a new node was only allowed if the bigger node contained at least 2 samples, and each resulting node at least 1. Gini impurity was used to measure the quality of each possible split when building the tree, with node not being allowed to split again if their impurity was under $1 \cdot 10^{-7}$. To improve training times, the number of features that were considered for each split was reduced to $\sqrt{|F|}$, with $|F|$ being the total number of features. The number of trees trained for each Random Forest classifier is 10. To create a sample of the training data for each tree, bootstrap samples were used: instances were sampled with replacement, and thus had some chance of occurring multiple times in a single training sample.

For the Support Vector Machine models, the hyperparameters were set as follows. A linear kernel was used for the mapping of feature vectors. The penalization in the learning process was done combining L_2 regularization R and a squared hinge loss function (L), with the standard penalty parameter P set to 1: the goal is to minimize total loss $P \sum_{i=1}^{n} L(f(x_i), y_i) + R(w)$. Intercept is calculated, but intercept scaling is set in such a manner that the regularization does not affect this value as much, with a value of 1.

XGBoost again, has multiple parameters to define the boundaries for building the regression trees. The maximum tree depth is 3, with a number of 100 trees in the whole classifier. The minimum reduction of the loss function for a new split is set to 0: a split should not increase loss. The loss function uses L2 regularization. The problem is set to binary classification, with tree boosting and a learning rate of 0.1.

The other boosting algorithm, AdaBoost, was mostly dependent on the parameters of its base classifiers. The number of base classifiers or estimators was 50. The learning rate, which decreases the contribution of the estimators, was 1. SAMME.R boosting was used, which means that the predicted class probabilities are used to adapt the model, instead of the errors.

4.2 Evaluation measures

The values given in the tables are F-scores, unless otherwise indicated. Binary classifier results are usually given as four numbers, each for the number of test instances that are classified in a certain manner. True positives (TP) are the instances that were correctly classified as belonging to the interesting class, false positives (FP) are those that were incorrectly classified as such, and true negatives (TN) and false negatives (FN) are similar for instances not classified as belonging to the interesting class. These numbers can be used to calculate values that give some measure of how well a classifier performs. Accuracy shows the proportion of the test set that was classified correctly, precision shows what proportion of those instances that were classified as interesting actually are interesting and recall measures what proportion of the interesting instances in the test set were actually classified as such. F-score (or F-measure) is the harmonic mean of precision and recall. Below the definitions of these measures are given.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Precision and recall are usually considered to be more informative than accuracy, since these measures contain information on what type of misclassification has occurred and how often. This aspect is even more important when dealing with imbalanced data, as is the case in this project. Majority vote classification would give an accuracy of over 90% for general online crime, even though this type of classification is not informative at all. Since F-score is the harmonic mean of precision and recall, en thus provides some insight in how well the different types of misclassification are balanced, F-score was used as the measure of performance. An F-score approaching 1 means every classification in the test set is correct, while an F-score approaching 0 would mean that no True Positives were found during classification.

All results shown are F-scores based on the average scores of the model in 10-fold crossvalidation, unless otherwise indicated. This means that to calculate each score, ten classifiers were trained on different folds of the data. These were scored, and their numbers for TP, FP, TN, FN were averaged. These averages were used to calculate the F-score.

An additional evaluation measure is the degree of overfitting of a model. Overfitting happens when a model is too specifically tailored for the training data. The model then takes features into account that might work for the training set, but not quite as well on generalized, previously unseen data. When a model is overfit, a model performs better on the training set than on a test set. It might be the case that the best model is still somewhat overfit. However, overfitting should be avoided if possible, since it usually means the model is not general enough.

4.3 Results

Table 4.1 shows the F-scores for the five best scoring classification models for each of the four binary categories. The highest score for each binary class is in bold print. The F-scores for the other algorithms with the same preprocessing and boosting are included as well, to provide some sense of the range in which the different algorithms score in the same context. Table 4.3 shows more detailed information about the performance of the best model for each binary classification category, and gives some extra information about the ten most relevant features for classification. For general online crime, the top performing model is a SVM that used unigrams and bigrams, and bootstrapping resampling. The feature vectors were created using basic tf-idf values, and the

	Filter	**N-grams**	**Resampling**	**Boosting**	**TF-idf**	**Binary**	**Naive Bayes**	**Random Forest**	**SVM**	**XGBoost**
General online crime	Yes	1,2	none	No	No	No	0.603	0.276	0.668	0.488
			SMOTE	No	Yes	No	0.634	0.348	0.680	0.551
			Bootstrapping	No	Yes	No	0.645	0.429	**0.696**	0.570
						Yes	0.658	0.404	0.668	0.600
			ENN	No	No	Yes	0.630	0.296	0.690	0.439
			SMOTE-ENN	No	No	Yes	0.594	0.372	0.691	0.522
Online threat	Yes	1	none	No	No	No	**0.696**	0.118	0.497	0.440
						Yes	0.689	0.134	0.447	0.417
			ENN	No	No	No	0.694	0.117	0.480	0.413
						Yes	0.693	0.119	0.468	0.426
			SMOTE-ENN	No	No	No	0.478	0.473	0.682	0.540
Online distribution of sexually obscene imagery	Yes	1	none	No	No	No	0.667	0.082	0.454	0.388
			Bootstrapping	No	Yes	Yes	0.301	0.362	0.205	0.723
						No	0.327	0.449	0.222	0.691
			ENN	No	No	Yes	0.667	0.128	0.413	0.444
		1,2	none	No	No	Yes	**0.725**	0.000	0.444	0.426
			ENN	No	No	Yes	0.724	0.029	0.409	0.400
Computer trespass	Yes	1	none	No	Yes	Yes	0.598	0.274	0.274	0.537
			ENN	No	Yes	Yes	0.605	0.274	0.303	0.513
		1,2	Bootstrapping	No	Yes	Yes	0.427	0.291	0.211	0.594
			ENN	No	No	No	0.423	0.091	0.581	0.447
			SMOTE-ENN	No	No	Yes	0.274	0.116	**0.616**	0.524

Table 4.1: F-scores of the top-5 best-performing models for all four binary categories

F-score of the model was 0.696. As can be seen in table 4.3, the accuracy for this model was the lowest of the top models, with 94% accuracy. The model is also very likely overfit, since there is a large difference between the F-scores for the training set and test set.

The best model for online threat scored the same, 0.696. It used basic word count vectors of unigrams to train a Naive Bayes classifier without resampling algorithms. This model seems to have the least degree of overfitting, since the difference between training set and test set F-scores is only 0.112.

The best model for online distribution of sexually obscene imagery had the highest F-score overall, 0.725, and the highest accuracy of 98.7%. This was surprising, since the models for this class often had no true positives (and thus an F-score of 0) in the first phase of the project. This Naive Bayes classifier used unigrams and bigrams to create binary term frequency vectors without resampling.

For computer trespass the best model was an SVM with an F-score of 0.616, meaning that is scores a bit lower than the best models for the other classes. This one also used binary term counts of unigrams and bigrams, and SMOTE resampling.

All of the four best-performing models used the term filtering and no boosting algorithm. The results shown in this section are only a selection of the full experimental results. The full results can be found in appendix section B.

	Online threat	Online distribution of obscene imagery	Computer trespass	All online crime
F-score	0.113	0.042	0.049	0.171

Table 4.2: F-scores for random classification for each class

Category	Accuracy	Precision	Recall	F-score (test set)	F-score (training set)	Most relevant features
General online crime	0.940	0.735	0.661	0.696	0.999	gebruiken, boos, aangifte, computervredebreuk, bedreiging, met, krijgen, naaktfoto, bedreigen, via
Online threat	0.955	0.614	0.803	0.696	0.808	hoerenkind, tijdlijn, uitspoken, verdiepen, wedden, whatapp, kk, whatsap, zometeen, tweets
Online distribution of sexually obscene imagery	0.987	0.667	0.79	0.725	0.864	verwijderen jou, via in, een naakt, jou haar, naaktfoto zijn, van kinderporno, zijn verspreiden, de naaktfoto, getint foto, kinderpornografisch
Computer trespass	0.977	0.543	0.713	0.616	0.820	overmaken, zij, vies, computervredebreuk, bedreiging, ongeveer, aangifte, via, de, hacken

Table 4.3: Scoring information and top-10 most relevant features for classification for best model for each category

4.3.1 Analysis

To start, we compare the F-scores in the table with the estimated scores of random classification[1] shown in table 4.2.
For all binary classification tasks, the best models have a performance of 3.5-11 times the random classification, which is a very large increase. The highest F-scores for all four models are in the range 0.6-0.75. While these numbers indicate that there is still some amount of data that is classified incorrectly by the models, the scores and the improvements from random show clearly that the information needed for the classification can be extracted from the text documents in the form of features. It might well be possible that further improvements can be made to increase the reliability of the classifiers.

Table 4.3 shows the most relevant features for classification for the best model of each binary category. It is interesting to see that many of those are very intuitive: "hoerenkind" (literally 'whore's child') is relevant for classification of online threat, "de naaktfoto" (nude picture) is relevant for classification of online distribution of sexually obscene imagery, and "hacken" (hacking) for computer trespass. Some are less obvious to see: for example "wedden" (to bet) is probably deemed relevant for classification of online threat since in police reports it is often used in a context like "He said: 'You don't believe I can kill you? Wanna bet?"'. It is good to note that this are the terms that are seen as relevant for determining the class. This means that these terms can occur relatively often in the interesting class, *or* in the uninteresting class. This explains the presence of terms like "ongeveer" (more or less, approximately) and "met" (with).

These relevant features also show a new factor that might have had a large influence on the model

[1]Giving each instance a random label with equal weight for each class. For binary classification this means that the distribution over the two classes would be approximately 50/50. Half of the positives would be classified correctly, half incorrectly, with the same holding for the negatives.

performance: spelling. Terms like “whatapp” and “whatsap” both obviously refer to Whatsapp. However, they are treated as separate terms by the model, which means they don’t increase the term frequency of the term “Whatsapp”. Taking measures to somehow connect these terms, as well as synonyms, might improve the model performance and decrease the number of features.

It is possible to calculate the most relevant features for getting the classification for each instance along with the class label. These features are calculated by getting the feature probabilities given a class for all features. The 10 features[2] with the highest absolute feature probabilities that occur in the document are estimated to be the most relevant for the classification. This means that both the features that have high probability for the positive and the negative class are included for any instance, since these both impact the final classification. This information can be very useful to provide some justification for the classification given by the model, as the presence of key words is something that is both easily checked and easily grasped by humans.

Preprocessing comparison

When we look at all models represented in table 4.1, there are some properties that they have in common. All of these models work with filtered data, which means that there has already been a pre-selection made on the terms in the documents that are used for creating the model and classifying new instances. A possible explanation for this lies in the dimensionality of the data. A text corpus of the size of the training set has a large lexicon. Since each term is a feature, the number of features can be extremely high, with most features having very low values for almost all instances. These features are removed when the filter is applied, which means all features that are left have a relatively high chance of being relevant for the classification.

Such an obvious explanation is not as easily found for the fact that all winning models make no use of AdaBoost boosting. Since the strength of AdaBoost is the fact that it combines multiple weaker models into a single stronger classifier, it seems strange that a single model is less accurate in its classification than the combination of several of those same models.

The full results for the general online crime class are summarized in figure 4.1. Similar graphs for the other 3 classes can be found in appendix section C. These graphs give an overview the average performance of the algorithms, and of how the preprocessing steps influence the performance. These graphs are purely meant as an indication: it is very well possible that a combination of steps that each perform below average can be a combination leading to a very good classifier. A good example is found in the graph comparing the results of tf-idf values to those that are based on word frequencies (figure 4.1, bottom left). The graph shows a very large difference in average performance between Naive Bayes models that do and do not use tf-idf values. However, when we look at the best-performing models, there are multiple models in which Naive Bayes is coupled with tf-idf feature vectors, while getting good scores. It is dangerous to draw strong conclusions from these graphs about what works and what does not. However, it can be very interesting to see what general ideas can be seen in them. When we look at the same graph for the other categories in appendix section C, we see that for those categories the same holds for SVMs. The scores for models based on word frequencies on average perform better than those of models based on tf-idf values.

[2]Often there are less than 10 for filtered data, since the most important features of a model will not occur in every document, and filtered data has relatively few features to begin with.

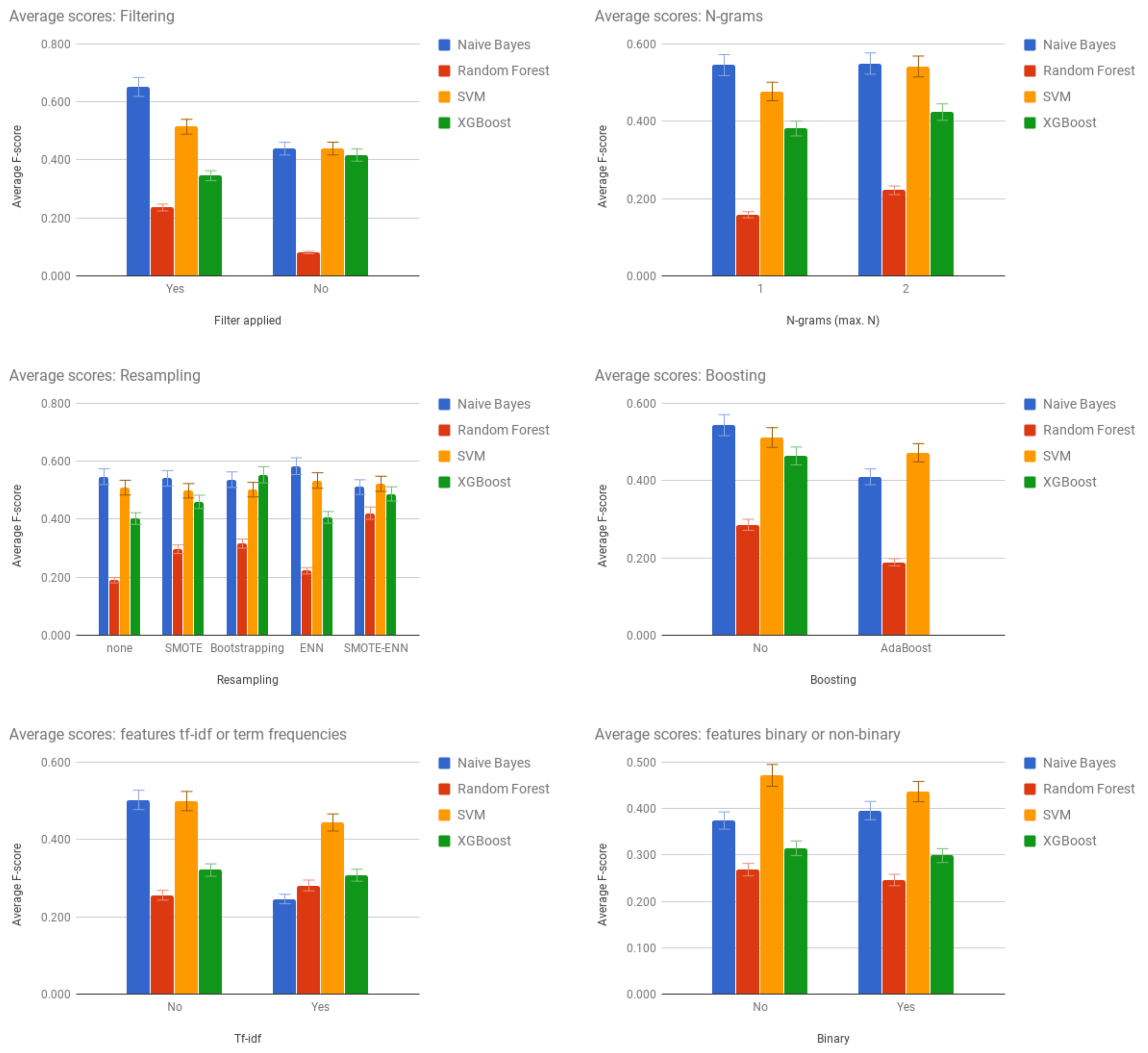

Figure 4.1: Graphs comparing the scores of different models and model choices for the general online crime category

The graphs also show that the Random Forest classifiers on average perform a lot worse than the models trained on other algorithms, independently from what preprocessing step is being examined. This could be an indication that this algorithm is not a good fit for the data. An explanation could be that Random Forest classifiers use a relatively small amount of features in the final classification process. It might be the case that this number of features is too little to completely cover what characterizes the interesting class, especially since the data contains such large amounts of features.

While some of the graphs do not show very large differences between options, there are a few interesting points. Firstly, applying the filter to decrease the amount of features significantly increases the average scores. The reason for this is probably that it removes noise: text data naturally contains large amounts of words that have appeared once in the lexicon, and never in multiple documents. These words carry very little information information about the characteristics of the class, and so can be removed to get a feature space that is more easily navigated during the learning process.

Of the four machine learning algorithms, Random Forest and XGBoost seems to benefit most from choosing the right resampling method. Especially the application of SMOTE-ENN resampling to a Random Forest classifier, seems to cause a spectacular increase in performance compared to other sampling methods. For Naive Bayes and SVM classifiers, the average score seems to be less dependent on the resampling algorithm: the average performance is comparable over the different methods. The difference in performance between the different resampling methods is slightly larger for the other three categories than for the general online crime class. This can easily be explained by the fact that the resampling has more influence in those classes, since the training data was more imbalanced for these categories than for the general online crime category.

The application of AdaBoost boosting did not have a large positive influence on the performance of the models, as can be seen in figure 4.1. This may be due to the noise in the data: since the data is relatively noisy, and has many outliers, it is possible that AdaBoost overfits on these instances by assigning them high weights during the training process. This way the boosting reinforces the mistakes of the base classifiers, instead of their correct classifications.

An interesting observation that can be seen from looking at the complete results in appendix section B is that Random Forest classifiers usually seem to profit from the use of tf-idf values instead of raw (binary) term frequencies. This is especially true when these classifiers are combined with AdaBoost boosting. In the cases where the data was filtered before training, almost all scores of Random Forest classifiers that combined tf-idf values and AdaBoost are significantly better than their non-tf-idf counterparts. For Naive Bayes and SVM classifiers, using TF-idf values instead of term frequencies only seemed to lower the average score.

Preliminary results

Halfway through this project, some classifiers were trained on only part of the data (1896 instances out of the total of 3096). The selection of algorithms and preprocessing steps in this phase of the experiment was much smaller. The results of this preliminary phase will not be discussed in depth, as the second phase contained models that performed much better overall. The full preliminary results can be found in the appendix, in table B.9. However, it is interesting to look at the difference that the increased amount of training data made.

	Filter	N-grams	Resampling	Boosting	TF-idf	Binary	Naive Bayes	Random Forest	SVM
General online crime	Yes	1	none	No	Yes	No	0.372	0.228	0.396
							0.287	0.181	0.376
						Yes	0.371	0.123	0.363
							0.282	0.140	0.359
				AdaBoost	Yes	No	0.375	0.492	0.421
							0.305	0.459	0.378
						Yes	0.383	0.426	0.389
							0.295	0.443	0.391
			SMOTE-ENN	No	Yes	No	0.236	0.237	0.236
							0.277	0.341	0.372
	No	1	none	No	Yes	No	0.256	0.087	0.252
							0.000	0.133	0.381
						Yes	0.167	0.076	0.211
							0	0.129	0.286
Online threat	Yes	1	none	No	Yes	No	0.220	0	0.200
							0.114	0.095	0.179
						Yes	0.188	0.015	0.176
							0.123	0.076	0.176
				AdaBoost	Yes	No	0.235	0.177	0.199
							0.163	0.447	0.180
						Yes	0.201	0.204	0.198
							0.144	0.462	0.180
			SMOTE-ENN	No	Yes	No	0.156	0.188	0.156
							0.233	0.336	0.349
	No	1	none	No	Yes	No	0.162	0.015	0.211
							0	0.076	0.340
						Yes	0.045	0	0.074
							0	0.010	0.161
Online distribution of sexually obscene imagery	Yes	1	none	No	Yes	No	0.083	0.046	0.122
							0.419	0.057	0.000
						Yes	0.085	0	0.083
							0.437	0.081	0.027
				AdaBoost	Yes	No	0.125	0	0.083
							0.031	0.187	0.000
						Yes	0.043	0	0.042
							0.087	0.111	0.027
			SMOTE-ENN	No	Yes	No	0.177	0.144	0.177
							0.198	0.376	0.322
	No	1	none	No	Yes	No	0	0	0
							0	0	0.156
						Yes	0	0	0
							0	0	0
Computer trespass	Yes	1	none	No	Yes	No	0.194	0.114	0.282
							0.581	0.232	0.289
						Yes	0.222	0.114	0.243
							0.598	0.274	0.274
				AdaBoost	Yes	No	0.243	0.225	0.243
							0.265	0.182	0.289
						Yes	0.267	0.200	0.270
							0.284	0.159	0.271
			SMOTE-ENN	No	Yes	No	0.175	0.210	0.176
							0.202	0.321	0.295
	No	1	none	No	Yes	No	0.171	0.031	0.197
							0	0.093	0.178
						Yes	0.061	0.031	0.031
							0	0.071	0.116

Table 4.4: Comparison of F-scores for preliminary results (white, 1896 training instances) and final results (grey, 3096 training instances).

Table 4.4 contains most of the results of the preliminary phase (white background), together with the respective results from the second phase (grey background) for comparison. At first glance, the results of the extra data seem unimpressive: many of the differences are small enough that they may be due to differences in the exact distribution over the folds for crossvalidation. Some of the results are even a little worse for the full training set than for the smaller data set. When we look at the lower half of the table, however, the differences are much more impressive. Especially the Naive Bayes classifiers without boosting show large improvements, scoring 3-5 times as high as with the smaller training set.

As has been explained in section 2.1, the last two classes, *Online distribution of sexually obscene imagery* and *Computer trespass* are very small. The amount of positively labeled instances for these two classes is low: only a couple of instances from the data set were labeled as belonging to these classes. The results may be an indication that while the amount of instances labeled as belonging to the classes *General online crime* and *Online threat* was adequate for training a model, the same did not hold for the remaining classes. These profited in a large degree from the increase in training data. This gives a good idea of the importance of data set size and balance: if there are not enough positive instances, it is nearly impossible for a machine learning algorithm to adequately generalize and model what the relevant features are.

A related note on the preliminary results in appendix B: the F-score of many of the classifiers for online distribution of sexually obscene images is 0. This can most likely be blamed on the very low percentage (2.27%) and count (43) of positive training data for this class in the first phase. With this little data to use in training, the chance of finding a correct generalization is very slim. As we can see from the new results, this was improved by increasing the amount of data.

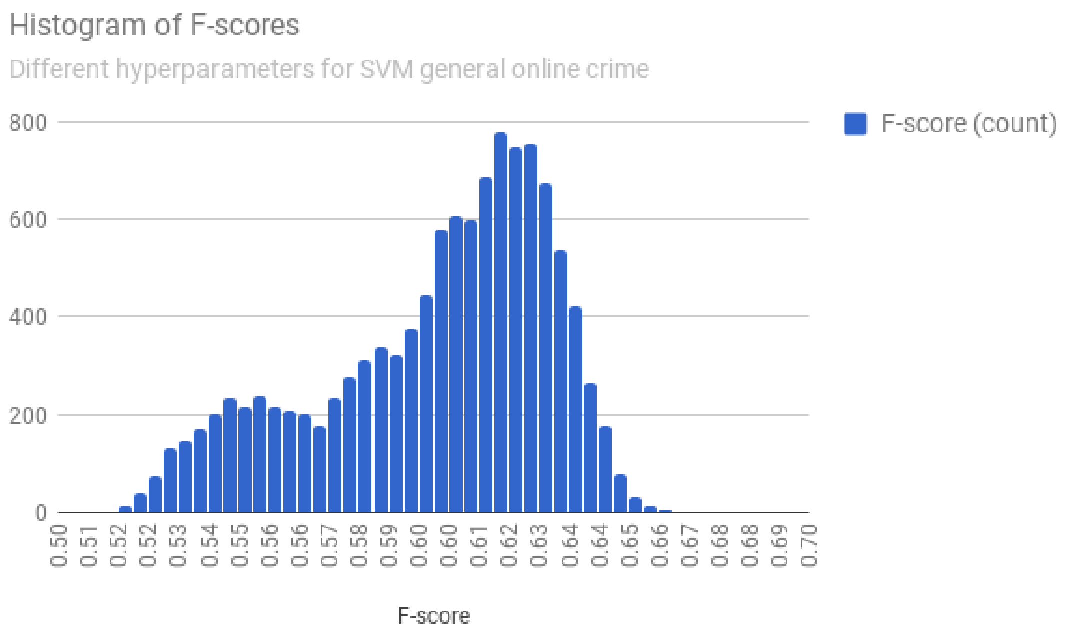

Figure 4.2: Histogram showing the F-scores for different parameter combinations for the best performing model of the general online crime classification

Parameter tuning

To try and find the optimal set of values of the hyperparameters of the best performing model, many different combinations of values were considered. For the best-performing model for the general online crime class, the results are summarized in figure 4.2. The earlier score of the model was an F-score of 0.696. As the histogram shows, none of the models with alternative parameter values had a score over 0.67. Tuning results for other models were similar to these. Further research is needed to determine the cause of this low performance, and to find an optimal hyperparameter configuration for this data.

Chapter 5

Conclusion

5.1 Discussion

In this thesis, I tried to find an answer to the question how text mining can be used to classify police reports. To make this question concrete, the goal was to train a model based on text mining for correct classification of three classes of online crime: online threat, online distribution of sexually obscene imagery, and computer trespass. A fourth umbrella class of general online crime was added as well. To this end different approaches for building a model were compared. Preprocessing steps and model options included linguistic preprocessing, multiple methods of feature construction and selection, boosting and resampling. Four different base algorithms were compared: Naive Bayes, Random Forest, SVM and XGBoost.

The experiment resulted in models with F-score of over 0.6 for all four binary categories. The best-performing model across all categories had an F-score of 0.725. All top-5 scores were great improvements over random classification, with F-scores of 3.5-11 times as high. The binary classifiers for the different classes can be combined to create a multiclass classifier for the complete classification task.

In this project, boosting (in the form of AdaBoost), and resampling (multiple algorithms) seemed to have very limited effect on the performance of the different classifiers. This might be due to conditions that can be improved, such as the amount of data, but this cannot be stated with certainty. Other factors, such as filtering out terms of low relevance, had a large positive impact on the model performances. However, more research on different data sets is needed to make any general claims.

The results of this research project are very promising, and they show that using text mining to classify police data is a very feasible option. However, the data set used in this project is small and the imbalance of the classes is very large. This causes problems for the classification, since it becomes difficult to generalize is there are few positive examples. Repeating the experiment with a larger training set would give more robust and reliable results. When increasing the training set, it might be advisable to have multiple experts annotate the data, to decrease possible ambiguity and ensure clear, correct annotation.

The impact that tuning the models has on model performance has not been a focus in this project. However, it is likely that using hyperparameters that are suited to the data will improve the performance of the classifiers. Ideally, the testing would have covered less algorithms, with a more in-depth analysis on which parameter values created a model most suited to the data set. This is especially true for the XGBoost models, which are known to rely heavily upon tuning for their performance. With that in mind, the average performance of the XGBoost models in this project is already quite impressive, but it could probably be improved. Other algorithms would most likely also profit from more consideration in this area.

5.1.1 Future research

As mentioned above, one of the main disadvantages of this project was the training set size. Since it is impossible to label all data, semi-supervised learning algorithms would be a good method to use the unlabeled data for training as well. This project did not have enough computational capacity to experiment with these algorithms, but with the large amount of unlabeled data in this set, chances of improvement are reasonably high.

Boosting algorithms usually increase performance. In this study, boosting results were not very good: many results actually got worse when boosting was applied. A worthwhile option would be to include more text-specific boosting algorithms, to see whether these would perform better than AdaBoost for this specific data set. More tuning on the (boosting) algorithms might also improve results. In this project, there was not enough focus on finding the optimal parameter values for each algorithm, which could have a large impact on the results.

Another step that has not been included, but which could have significant impact on the performance, is term normalization. Written text contains many synonyms, spelling errors and alternatives. In this projects, these were all counted as separate features, which means the total term frequency of those features is lower. Implementing some type of method to combine multiple spellings and synonyms into a single term, comparable to what lemmatization does, might make it easier for a model to discern relevant features. Additionally, this could greatly decrease the number of features, which could improve memory- and computational requirements, making it easier and faster to train the models.

In general, feature selection and reduction is seen by some as one of the most important steps of data mining. In this thesis, feature selection and reduction was limited to removing terms with certain PoS tags and the filtering of seemingly irrelevant terms by using tf-idf values. It would be interesting to look at other method of feature selection, including removing more stop words, more PoS tags and feature reduction algorithms such as Principal Component Analysis or feature clustering.

Bibliography

Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM Sigkdd Explorations Newsletter*, 6(1):20–29, 2004.

Rogerio Bonatti, Arthur G de Paula, Victor S Lamarca, and Fabio G Cozman. Effect of part-of-speech and lemmatization filtering in email classification for automatic reply. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Jingnian Chen, Houkuan Huang, Shengfeng Tian, and Youli Qu. Feature selection for text classification with naïve bayes. *Expert Systems with Applications*, 36(3):5432–5435, 2009.

Jinxiu Chen, Donghong Ji, Chew Lim Tan, and Zhengyu Niu. Relation extraction using label propagation based semi-supervised learning. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 129–136. Association for Computational Linguistics, 2006.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Ronen Feldman and James Sanger. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press, 2007.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. In *Ldv Forum*, volume 20, pages 19–62, 2005.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137–142, 1998.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Feng Kang, Rong Jin, and Rahul Sukthankar. Correlated label propagation with application to multi-label learning. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1719–1726. IEEE, 2006.

Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on knowledge and data engineering*, 17(4):491–502, 2005.

Julián Luengo, Alberto Fernández, Salvador García, and Francisco Herrera. Addressing data complexity for imbalanced data sets: analysis of smote-based oversampling and evolutionary undersampling. *Soft Computing*, 15(10):1909–1936, 2011.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *An Introduction To Information Retrieval*, 151:177, 2008.

Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Madison, WI, 1998.

Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. *Machine learning*, 39(2-3):103–134, 2000.

Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

Robert E Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine learning*, 39(2):135–168, 2000.

Bo Tang, Haibo He, Paul M Baggenstoss, and Steven Kay. A bayesian classification approach using class-specific features for text categorization. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1602–1606, 2016.

A. Van den Bosch, G.J. Busser, W. Daelemans, and S Canisius. An efficient memory-based morphosyntactic tagger and parser for dutch. In F. van Eynde, P. Dirix, I. Schuurman, and V. Vandeghinste, editors, *Selected Papers of the 17th Computational Linguistics in the Netherlands Meeting, Leuven, Belgium*, pages 99–114, 2007.

Maarten van Gompel, Ko van der Sloot, and Antal van den Bosch. Ucto: Unicode tokeniser. Technical report, Reference Guide, Technical report, Tilburg Centre for Cognition and Communication, Tilburg University and Radboud Centre for Language Studies, Radboud University Nijmegen. http://ilk. uvt. nl/downloads/pub/papers/ilk. 1205. pdf, 2012.

Shi-jin Wang, Avin Mathew, Yan Chen, Li-feng Xi, Lin Ma, and Jay Lee. Empirical analysis of support vector machine ensemble classifiers. *Expert Systems with applications*, 36(3):6466–6476, 2009.

Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(3):408–421, 1972.

Yonghua Xie, Yurong Liu, and Qingqiu Fu. Imbalanced data sets classification based on svm for sand-dust storm warning. *Discrete Dynamics in Nature and Society*, 2015, 2015.

Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. *Citeseer*, 2002.

Appendix A

Query

Listing A.1: Query for retrieving cybercrime data from the database Basis Voorziening Handhaving, Dutch National Police

```
contains\_oracle([Incidenten].[Product zoeken (incident inner join)].[
    Incident productinhoud] ;  '(Whatsapp␣or␣twitter␣or␣facebook␣or␣
    instagram␣or␣snapchat␣or␣4chan␣or␣skype␣or␣pinterest␣or␣wechat␣or␣
    ask.fm␣or␣kik␣or␣dumpert␣or␣messenger␣or␣msp␣or␣moviestarplanet␣or␣
    habbo␣or␣gosupermodel␣or␣youtube␣or␣momio␣or␣minecraft␣or␣tinder␣or␣
    webcam␣or␣camera␣or␣{world␣of␣warcraft}␣␣or␣gsm␣or␣mobiel␣or␣
    mobieltje␣or␣telefoon␣or␣foto␣or␣filmpje␣or␣account␣or␣accountnaam␣
    or␣profielfoto␣or␣chatten␣or␣nepaccount␣or␣{ip−adres}␣or␣game␣or␣
    xbox␣or␣playstation␣or␣meme␣or␣website␣or␣platform␣or␣online␣or␣
    computer)␣and␣(dreigen␣or␣␣bedreigen␣or␣dwang␣or␣chantage␣or␣dood␣or
    ␣dwingen␣or␣druk␣or␣afpers␣or␣laster␣or␣radicalisering␣or␣{haat␣
    zaaien}␣␣or␣{SR␣284/1}␣or␣{SR␣285/1}␣or␣{SR␣318/1})␣'  ; 1 ) > 0
```

or

```
contains\_oracle([Incidenten].[Product zoeken (incident inner join)].[
    Incident productinhoud] ;
'(Whatsapp␣or␣twitter␣or␣facebook␣or␣instagram␣or␣snapchat␣or␣4chan␣or␣
    skype␣or␣pinterest␣or␣wechat␣or␣ask.fm␣or␣kik␣or␣dumpert␣or␣
    messenger␣or␣msp␣or␣moviestarplanet␣or␣habbo␣or␣gosupermodel␣or␣
    youtube␣or␣momio␣or␣minecraft␣or␣tinder␣or␣webcam␣or␣camera␣or␣{
    world␣of␣warcraft}␣␣or␣gsm␣or␣mobiel␣or␣mobieltje␣or␣telefoon␣or␣
    foto␣or␣filmpje␣or␣account␣or␣accountnaam␣or␣profielfoto␣or␣chatten␣
    or␣nepaccount␣or␣{ip−adres}␣or␣game␣or␣xbox␣or␣playstation␣or␣meme␣
    or␣website␣or␣platform␣or␣online␣or␣computer)␣and␣(Cracken␣or␣
    Cracking␣or␣Hacken␣or␣hacking␣or␣inloggen␣or␣identiteitsfraude␣or␣
    wachtwoord␣or␣nepprofiel␣or␣credits␣or␣username␣or␣password␣or␣
    gebruikersnaam␣or␣wachtwoord␣or␣spionage␣or␣sabotage␣or␣{black␣hat}␣
    or␣{grey␣hat}␣or␣{white␣hat}␣or␣computervredebreuk␣or␣ransomware␣or␣
    exploit␣or␣{Angler␣Nuclear}␣or␣RIG␣or␣Magnitude␣or␣Neutrino␣or␣{
    Sweet␣Orange}␣or␣Fiesta␣or␣CK␣or␣Sundown␣or␣Blackhole␣or␣cryptoware␣
    or␣malware␣or␣{bit␣crypter}␣or␣cryptolocker␣or␣cryptowall␣or␣{CTB−
    Locker}␣or␣Locky␣or␣{Cybercrime−as−a−service}␣or␣Booterservice␣or␣
    DDos␣or␣KeRanger␣or␣Server␣or␣Nemucod␣or␣Encryptoraas␣or␣Crowti␣or␣
    Coinvault␣or␣Alphacrypt␣or␣Teslacrypt␣or␣{Watering−hole}␣or␣
    Infecteren␣or␣Jailbreak␣or␣Acedeceiver␣or␣{Flash␣Player}␣or␣{Zero␣
    days}␣or␣Trojan␣or␣{Remote␣Access␣Tool}␣or␣Phishing␣or␣
    Amplificatieaanval␣or␣{USB−thief}␣or␣{Cherry␣Picker}␣or␣Duqu␣or␣
    Doxing␣or␣Cybervanda\%␣or␣Scriptkiddies␣or␣grooming␣or␣ransomware␣or
    ␣roqueware␣or␣sofwarerobots␣or␣BOT␣or␣virus␣or␣defacing␣or␣cyber\%␣
    or␣{SR␣138A\%}␣or␣{SR␣139C/1}␣or␣{SR␣139D/\%})␣␣'   ; 2 ) > 0
```

or

```
contains\_oracle([Incidenten].[Product zoeken (incident inner join)].[
    Incident productinhoud] ;  '(Whatsapp␣or␣twitter␣or␣facebook␣or␣
    instagram␣or␣snapchat␣or␣4chan␣or␣skype␣or␣pinterest␣or␣wechat␣or␣
    ask.fm␣or␣kik␣or␣dumpert␣or␣messenger␣or␣msp␣or␣moviestarplanet␣or␣
    habbo␣or␣gosupermodel␣or␣youtube␣or␣momio␣or␣minecraft␣or␣tinder␣or␣
    webcam␣or␣camera␣or␣{world␣of␣warcraft}␣␣or␣gsm␣or␣mobiel␣or␣
    mobieltje␣or␣telefoon␣or␣foto␣or␣filmpje␣or␣account␣or␣accountnaam␣
    or␣profielfoto␣or␣chatten␣or␣nepaccount␣or␣{ip−adres}␣or␣game␣or␣
    xbox␣or␣playstation␣or␣meme␣or␣website␣or␣platform␣or␣online␣or␣
    computer)␣and␣(naaktfoto␣or␣naaktfilmpje␣or␣seksfilmpje␣or␣sexting␣
    or␣sextortion␣or␣sexchatting␣or␣selfie␣or␣sexy␣or␣bloot␣or␣borsten␣
    or␣borst␣or␣piemel␣or␣penis␣or␣vagina␣or␣bh␣or␣ondergoed␣or␣seksueel
    ␣or␣nude␣or␣kinderporno␣or␣smaad␣or␣laster␣or␣belediging␣or␣
    portretrecht␣or␣webcamseks␣or␣seks␣or␣␣sex␣or␣pose␣or␣porno␣or␣
    pornobeelden␣or␣seksplaatjes␣or␣{SR␣240/1}␣or␣{SR␣240A}␣␣or␣{SR␣
    261/1}␣or␣{SR␣262/1}␣)'   ; 3 ) > 0
```

Appendix B

Full experimental results

Filter	**N-grams**	**Resampling**	**Boosting**	**TF-idf**	**Binary**	**Naive Bayes**	**Random Forest**	**SVM**	**XGBoost**
Yes	1	none	No	No	No	0.652	0.236	0.515	0.346
					Yes	0.644	0.209	0.508	0.370
				Yes	No	0.287	0.181	0.376	0.407
					Yes	0.282	0.140	0.359	0.415
			AdaBoost	No	No	0.489	0.071	0.485	-
					Yes	0.493	0.083	0.476	-
				Yes	No	0.305	0.459	0.378	-
					Yes	0.295	0.443	0.391	-
		SMOTE	No	No	No	0.558	0.356	0.477	0.453
					Yes	0.564	0.351	0.484	0.417
				Yes	No	0.579	0.468	0.484	0.543
					Yes	0.590	0.476	0.470	0.560
			AdaBoost	No	No	0.486	0.214	0.486	-
					Yes	0.481	0.171	0.475	-
				Yes	No	0.323	0.505	0.505	-
					Yes	0.325	0.502	0.505	-
		Bootstrapping	No	No	No	0.539	0.392	0.500	0.523
					Yes	0.538	0.391	0.487	0.553
				Yes	No	0.588	0.558	0.510	0.579
					Yes	0.586	0.516	0.507	0.578
			AdaBoost	No	No	0.521	0.393	0.485	-
					Yes	0.493	0.319	0.509	-
				Yes	No	0.317	0.472	0.485	-
					Yes	0.297	0.486	0.490	-
		ENN	No	No	No	0.648	0.212	0.495	0.366
					Yes	0.646	0.215	0.492	0.351
				Yes	No	0.322	0.231	0.413	0.425
					Yes	0.313	0.173	0.390	0.420
			AdaBoost	No	No	0.490	0.100	0.495	-
					Yes	0.481	0.094	0.485	-
				Yes	No	0.325	0.384	0.411	-
					Yes	0.302	0.411	0.396	-
		SMOTE-ENN	No	No	No	0.536	0.447	0.565	0.487
					Yes	0.545	0.464	0.537	0.490
				Yes	No	0.277	0.341	0.372	0.349
					Yes	0.267	0.342	0.385	0.334
			AdaBoost	No	No	0.515	0.429	0.531	-
					Yes	0.524	0.459	0.534	-
				Yes	No	0.340	0.345	0.366	-
					Yes	0.349	0.342	0.363	-
	1,2	none	No	No	No	0.603	0.276	0.668	0.488
					Yes	0.629	0.246	0.662	0.433
				Yes	No	0.037	0.251	0.596	0.462
					Yes	0.193	0.227	0.579	0.419
		SMOTE	No	No	No	0.597	0.347	0.662	0.509
					Yes	0.605	0.238	0.646	0.462
				Yes	No	0.634	0.348	0.680	0.551
					Yes	0.658	0.440	0.650	0.556
		Bootstrapping	No	No	No	0.587	0.419	0.662	0.575
					Yes	0.585	0.406	0.667	0.602
				Yes	No	0.645	0.429	0.696	0.570
					Yes	0.658	0.404	0.668	0.600
		ENN	No	No	No	0.600	0.292	0.667	0.462
					Yes	0.630	0.296	0.690	0.439
				Yes	No	0.118	0.265	0.651	0.518
					Yes	0.239	0.274	0.618	0.446
		SMOTE-ENN	No	No	No	0.558	0.469	0.630	0.526
					Yes	0.594	0.372	0.691	0.522
				Yes	No	0.189	0.196	0.202	0.205
					Yes	0.188	0.189	0.189	0.190

Table B.1: Results for general online crime

Filter	**N-grams**	**Resampling**	**Boosting**	**TF-idf**	**Binary**	**Naive Bayes**	**Random Forest**	**SVM**	**XGBoost**
No	1	none	No	No	No	0.439	0.080	0.440	0.417
					Yes	0.154	0.086	0.438	0.385
				Yes	No	0.000	0.133	0.381	0.398
					Yes	0.000	0.129	0.286	0.377
		SMOTE	No	No	No	0.508	0.240	0.429	0.468
					Yes	0.480	0.091	0.418	0.405
				Yes	No	0.000	0.215	0.521	0.481
					Yes	0.036	0.196	0.421	0.458
		Bootstrapping	No	No	No	0.502	0.180	0.423	0.563
					Yes	0.489	0.202	0.435	0.571
				Yes	No	0.000	0.226	0.528	0.511
					Yes	0.025	0.171	0.423	0.522
	1,2	none	No	No	No	0.495	0.167	0.416	0.359
					Yes	0.489	0.161	0.376	0.342
				Yes	No	0.000	0.189	0.329	0.391
					Yes	0.000	0.137	0.217	0.370
			AdaBoost	No	No	0.178	0.031	0.440	-
					Yes	0.358	0.012	0.354	-
				Yes	No	0.041	0.012	0.314	-
					Yes	0.480	0.006	0.182	-
		SMOTE	No	No	No	0.501	0.244	0.426	0.409
					Yes	0.465	0.089	0.475	0.388
				Yes	No	0.120	0.156	0.422	0.451
					Yes	0.257	0.224	0.316	0.447
			AdaBoost	No	No	0.208	0.110	0.437	-
					Yes	0.418	0.012	0.427	-
				Yes	No	0.200	0.053	0.430	-
					Yes	0.393	0.134	0.316	-
		Bootstrapping	No	No	No	0.520	0.274	0.424	0.553
					Yes	0.475	0.267	0.354	0.512
				Yes	No	0.124	0.343	0.470	0.514
					Yes	0.248	0.219	0.308	0.486
			AdaBoost	No	No	0.391	0.164	0.418	-
					Yes	0.443	0.149	0.340	-
				Yes	No	0.441	0.159	0.450	-
					Yes	0.470	0.128	0.318	-
		ENN	No	No	No	0.5022222222	0.1628498728	0.4393673111	0.3918918919
					Yes	0.488946684	0.1322751323	0.3765690377	0.3551401869
				Yes	No	0	0.1855670103	0.3556581986	0.4044444444
					Yes	0	0.1832061069	0.2356020942	0.3686635945
		SMOTE-ENN	No	No	No	0.439	0.347	0.372	0.449
					Yes	0.216	0.192	0.197	0.199
				Yes	No	0.1880141011	0.1978227061	0.2155525239	0.238
					Yes	0.1880693506	0.1885900089	0.1888132584	0.188

Table B.2: Results for general online crime - part 2

Filter	N-grams	Resampling	Boosting	TF-idf	Binary	Naive Bayes	Random Forest	SVM	XGBoost
Yes	1	none	No	No	No	0.696	0.118	0.497	0.440
					Yes	0.689	0.134	0.447	0.417
				Yes	No	0.114	0.095	0.179	0.511
					Yes	0.123	0.076	0.176	0.519
			AdaBoost	No	No	0.453	0.086	0.475	-
					Yes	0.484	0.095	0.490	-
				Yes	No	0.163	0.447	0.180	-
					Yes	0.144	0.462	0.180	-
		SMOTE	No	No	No	0.457	0.297	0.472	0.515
					Yes	0.467	0.301	0.494	0.487
				Yes	No	0.500	0.506	0.340	0.568
					Yes	0.503	0.433	0.329	0.593
			AdaBoost	No	No	0.558	0.293	0.495	-
					Yes	0.489	0.211	0.472	-
				Yes	No	0.194	0.513	0.302	-
					Yes	0.225	0.517	0.317	-
		Bootstrapping	No	No	No	0.390	0.332	0.491	0.529
					Yes	0.370	0.401	0.480	0.526
				Yes	No	0.456	0.548	0.335	0.596
					Yes	0.457	0.510	0.317	0.586
			AdaBoost	No	No	0.519	0.338	0.500	-
					Yes	0.527	0.318	0.447	-
				Yes	No	0.171	0.508	0.313	-
					Yes	0.185	0.490	0.323	-
		ENN	No	No	No	0.694	0.117	0.480	0.413
					Yes	0.693	0.119	0.468	0.426
				Yes	No	0.123	0.152	0.197	0.510
					Yes	0.123	0.137	0.201	0.526
			AdaBoost	No	No	0.475	0.096	0.479	-
					Yes	0.403	0.104	0.445	-
				Yes	No	0.175	0.444	0.194	-
					Yes	0.164	0.473	0.197	-
		SMOTE-ENN	No	No	No	0.446	0.410	0.540	0.536
					Yes	0.460	0.434	0.534	0.539
				Yes	No	0.233	0.336	0.349	0.316
					Yes	0.237	0.327	0.345	0.307
			AdaBoost	No	No	0.544	0.375	0.552	-
					Yes	0.531	0.441	0.539	-
				Yes	No	0.237	0.322	0.349	-
					Yes	0.249	0.321	0.345	-
	1,2	none	No	No	No	0.557	0.157	0.634	0.476
					Yes	0.620	0.082	0.614	0.434
				Yes	No	0.000	0.076	0.464	0.495
					Yes	0.000	0.065	0.318	0.443
		SMOTE	No	No	No	0.503	0.299	0.641	0.529
					Yes	0.506	0.139	0.640	0.471
				Yes	No	0.581	0.257	0.599	0.560
					Yes	0.614	0.359	0.570	0.513
		Bootstrapping	No	No	No	0.508	0.379	0.647	0.576
					Yes	0.517	0.297	0.636	0.556
				Yes	No	0.608	0.390	0.627	0.591
					Yes	0.628	0.441	0.542	0.599
		ENN	No	No	No	0.552	0.144	0.624	0.510
					Yes	0.619	0.123	0.648	0.444
				Yes	No	0.000	0.171	0.551	0.531
					Yes	0.000	0.092	0.332	0.469
		SMOTE-ENN	No	No	No	0.478	0.473	0.682	0.540
					Yes	0.520	0.239	0.665	0.558
				Yes	No	0.121	0.148	0.148	0.156
					Yes	0.120	0.130	0.122	0.135

Table B.3: Results for online threat

Filter	N-grams	Resampling	Boosting	TF-idf	Binary	Naive Bayes	Random Forest	SVM	XGBoost
No	1	none	No	No	No	0.314	0.000	0.441	0.464
					Yes	0.047	0.057	0.380	0.373
				Yes	No	0.000	0.076	0.340	0.444
					Yes	0.000	0.010	0.161	0.362
		SMOTE	No	No	No	0.442	0.159	0.415	0.484
					Yes	0.378	0.048	0.358	0.423
				Yes	No	0.010	0.085	0.455	0.508
					Yes	0.048	0.119	0.328	0.500
		Bootstrapping	No	No	No	0.434	0.142	0.412	0.572
					Yes	0.403	0.132	0.393	0.513
				Yes	No	0.020	0.115	0.479	0.545
					Yes	0.020	0.090	0.306	0.519
	1,2	none	No	No	No	0.441	0.082	0.420	0.444
					Yes	0.395	0.029	0.327	0.360
				Yes	No	0.000	0.048	0.267	0.427
					Yes	0.000	0.048	0.059	0.385
			AdaBoost	No	No	0.073	0.010	0.453	-
					Yes	0.263	0.010	0.338	-
				Yes	No	0.048	0.000	0.244	-
					Yes	0.409	0.000	0.086	-
		SMOTE	No	No	No	0.433	0.159	0.458	0.447
					Yes	0.368	0.047	0.327	0.417
				Yes	No	0.117	0.115	0.426	0.509
					Yes	0.288	0.171	0.246	0.479
			AdaBoost	No	No	0.347	0.112	0.427	-
					Yes	0.317	0.000	0.311	-
				Yes	No	0.170	0.000	0.428	-
					Yes	0.217	0.063	0.239	-
		Bootstrapping	No	No	No	0.431	0.169	0.445	0.582
					Yes	0.407	0.278	0.353	0.519
				Yes	No	0.117	0.218	0.439	0.540
					Yes	0.225	0.145	0.208	0.471
			AdaBoost	No	No	0.389	0.188	0.456	-
					Yes	0.420	0.207	0.349	-
				Yes	No	0.406	0.232	0.442	-
					Yes	0.412	0.159	0.231	-
		ENN	No	No	No				
					Yes	0.4045584046	0.037037037	0.3247232472	0.3736263736
				Yes	No	0	0.049	0.270	0.471
					Yes	0	0.0921658986	0.0943396226	0.4217687075

Table B.4: Results for online threat - part 2

Filter	N-grams	Resampling	Boosting	TF-idf	Binary	Naive Bayes	Random Forest	SVM	XGBoost
Yes	1	none	No	No	No	0.667	0.082	0.454	0.388
					Yes	0.642	0.054	0.409	0.408
				Yes	No	0.419	0.057	0.000	0.490
					Yes	0.437	0.081	0.027	0.495
			AdaBoost	No	No	0.316	0.000	0.379	-
					Yes	0.344	0.000	0.449	-
				Yes	No	0.031	0.187	0.000	-
					Yes	0.087	0.111	0.027	-
		SMOTE	No	No	No	0.264	0.137	0.417	0.481
					Yes	0.260	0.108	0.392	0.538
				Yes	No	0.304	0.412	0.258	0.569
					Yes	0.312	0.431	0.182	0.632
			AdaBoost	No	No	0.364	0.133	0.426	-
					Yes	0.426	0.228	0.417	-
				Yes	No	0.158	0.348	0.196	-
					Yes	0.088	0.375	0.264	-
		Bootstrapping	No	No	No	0.176	0.175	0.429	0.569
					Yes	0.178	0.205	0.451	0.514
				Yes	No	0.327	0.449	0.222	0.691
					Yes	0.301	0.362	0.205	0.723
			AdaBoost	No	No	0.423	0.220	0.409	-
					Yes	0.328	0.198	0.429	-
				Yes	No	0.131	0.449	0.227	-
					Yes	0.109	0.469	0.222	-
		ENN	No	No	No	0.628	0.028	0.409	0.454
					Yes	0.667	0.128	0.413	0.444
				Yes	No	0.437	0.028	0.053	0.475
					Yes	0.455	0.029	0.000	0.475
			AdaBoost	No	No	0.250	0.000	0.396	-
					Yes	0.369	0.000	0.430	-
				Yes	No	0.087	0.057	0.027	-
					Yes	0.031	0.111	0.027	-
		SMOTE-ENN	No	No	No	0.256	0.156	0.430	0.534
					Yes	0.268	0.173	0.431	0.526
				Yes	No	0.198	0.376	0.322	0.346
					Yes	0.192	0.363	0.330	0.347
			AdaBoost	No	No	0.431	0.103	0.419	-
					Yes	0.369	0.028	0.434	-
				Yes	No	0.154	0.341	0.330	-
					Yes	0.160	0.359	0.343	-
	1,2	none	No	No	No	0.472	0.056	0.505	0.458
					Yes	0.725	0.000	0.444	0.426
				Yes	No	0.000	0.029	0.056	0.349
					Yes	0.000	0.029	0.000	0.396
		SMOTE	No	No	No	0.114	0.135	0.468	0.515
					Yes	0.044	0.029	0.376	0.412
				Yes	No	0.330	0.154	0.341	0.480
					Yes	0.265	0.230	0.111	0.537
		Bootstrapping	No	No	No	0.108	0.158	0.489	0.574
					Yes	0.031	0.104	0.432	0.500
				Yes	No	0.327	0.133	0.337	0.574
					Yes	0.318	0.378	0.083	0.640
		ENN	No	No	No	0.495	0.107	0.571	0.495
					Yes	0.724	0.029	0.409	0.400
				Yes	No	0.000	0.057	0.056	0.326
					Yes	0.000	0.029	0.000	0.454
		SMOTE-ENN	No	No	No	0.183	0.340	0.571	0.529
					Yes	0.103	0.104	0.479	0.441
				Yes	No	0.043	0.059	0.049	0.055
					Yes	0.043	0.051	0.043	0.051

Table B.5: Results for online distribution of sexual images

Filter	N-grams	Resampling	Boosting	TF-idf	Binary	Naive Bayes	Random Forest	SVM	XGBoost
No	1	none	No	No	No	0.044	0.000	0.214	0.302
					Yes	0.000	0.000	0.227	0.322
				Yes	No	0.000	0.000	0.156	0.286
					Yes	0.000	0.000	0.000	0.262
		SMOTE	No	No	No	0.339	0.053	0.229	0.455
					Yes	0.309	0.000	0.322	0.408
				Yes	No	0.082	0.025	0.431	0.422
					Yes	0.177	0.027	0.272	0.472
		Bootstrapping	No	No	No	0.338	0.000	0.242	0.455
					Yes	0.340	0.000	0.295	0.487
				Yes	No	0.081	0.000	0.396	0.487
					Yes	0.107	0.000	0.317	0.454
	1,2	none	No	No	No	0.254	0.029	0.239	0.364
					Yes	0.028	0.029	0.080	0.292
				Yes	No	0.000	0.029	0.108	0.286
					Yes	0.000	0.029	0.000	0.212
			AdaBoost	No	No	0.000	0.000	0.240	-
					Yes	0.177	0.000	0.056	-
				Yes	No	0.000	0.000	0.108	-
					Yes	0.000	0.000	0.000	-
		SMOTE	No	No	No	0.277	0.056	0.252	0.430
					Yes	0.234	0.000	0.083	0.304
				Yes	No	0.348	0.029	0.273	0.426
					Yes	0.316	0.027	0.000	0.419
			AdaBoost	No	No	0.175	0.028	0.214	-
					Yes	0.288	0.000	0.056	-
				Yes	No	0.187	0.000	0.311	-
					Yes	0.081	0.077	0.029	-
		Bootstrapping	No	No	No	0.247	0.056	0.214	0.504
					Yes	0.053	0.056	0.110	0.420
				Yes	No	0.196	0.000	0.311	0.475
					Yes	0.277	0.110	0.029	0.462
			AdaBoost	No	No	0.093	0.000	0.192	-
					Yes	0.189	0.000	0.057	-
				Yes	No	0.083	0.029	0.289	-
					Yes	0.103	0.057	0.000	-
		ENN	No	No	No				
					Yes	0.0833333333	0.000	0.056	0.203
				Yes	No	0	0.029	0.108	0.295
					Yes	0	0.0289855072	0	0.217
		SMOTE-ENN	No	No	No				
					Yes				
				Yes	No				
					Yes				

Table B.6: Results for online distribution of sexual images - part 2

Filter	N-grams	Resampling	Boosting	TF-idf	Binary	Naive Bayes	Random Forest	SVM	XGBoost
Yes	1	none	No	No	No	0.507	0.151	0.479	0.483
					Yes	0.523	0.245	0.456	0.466
				Yes	No	0.581	0.232	0.289	0.500
					Yes	0.598	0.274	0.274	0.537
			AdaBoost	No	No	0.369	0.118	0.458	-
					Yes	0.325	0.025	0.460	-
				Yes	No	0.265	0.182	0.289	-
					Yes	0.284	0.159	0.271	-
		SMOTE	No	No	No	0.371	0.263	0.426	0.496
					Yes	0.364	0.308	0.480	0.463
				Yes	No	0.414	0.508	0.376	0.564
					Yes	0.418	0.464	0.330	0.562
			AdaBoost	No	No	0.390	0.277	0.471	-
					Yes	0.368	0.222	0.446	-
				Yes	No	0.258	0.504	0.365	-
					Yes	0.256	0.557	0.407	-
		Bootstrapping	No	No	No	0.308	0.345	0.425	0.466
					Yes	0.313	0.291	0.512	0.449
				Yes	No	0.415	0.472	0.383	0.562
					Yes	0.417	0.526	0.393	0.531
			AdaBoost	No	No	0.310	0.308	0.410	-
					Yes	0.348	0.252	0.492	-
				Yes	No	0.269	0.535	0.356	-
					Yes	0.228	0.567	0.362	-
		ENN	No	No	No	0.496	0.172	0.414	0.448
					Yes	0.502	0.283	0.487	0.446
				Yes	No	0.569	0.217	0.271	0.517
					Yes	0.605	0.274	0.303	0.513
			AdaBoost	No	No	0.336	0.025	0.429	-
					Yes	0.323	0.049	0.456	-
				Yes	No	0.267	0.182	0.289	-
					Yes	0.292	0.182	0.289	-
		SMOTE-ENN	No	No	No	0.347	0.336	0.510	0.487
					Yes	0.353	0.272	0.510	0.458
				Yes	No	0.202	0.321	0.295	0.246
					Yes	0.198	0.317	0.301	0.250
			AdaBoost	No	No	0.395	0.345	0.446	-
					Yes	0.377	0.277	0.510	-
				Yes	No	0.233	0.333	0.288	-
					Yes	0.246	0.345	0.290	-
	1,2	none	No	No	No	0.430	0.182	0.574	0.427
					Yes	0.504	0.138	0.561	0.426
				Yes	No	0.000	0.115	0.182	0.431
					Yes	0.000	0.178	0.000	0.438
		SMOTE	No	No	No	0.389	0.132	0.339	0.409
					Yes	0.349	0.029	0.317	0.358
				Yes	No	0.345	0.151	0.364	0.446
					Yes	0.369	0.207	0.267	0.440
		Bootstrapping	No	No	No	0.268	0.190	0.562	0.543
					Yes	0.209	0.269	0.491	0.493
				Yes	No	0.411	0.130	0.446	0.550
					Yes	0.427	0.291	0.211	0.594
		ENN	No	No	No	0.423	0.091	0.581	0.447
					Yes	0.510	0.161	0.544	0.427
				Yes	No	0.000	0.140	0.274	0.392
					Yes	0.000	0.118	0.000	0.369
		SMOTE-ENN	No	No	No	0.286	0.397	0.475	0.484
					Yes	0.274	0.116	0.616	0.524
				Yes	No	0.050	0.057	0.051	0.060
					Yes	0.049	0.053	0.050	0.055

Table B.7: Results for computer trespass

Filter	N-grams	Resampling	Boosting	TF-idf	Binary	Naive Bayes	Random Forest	SVM	XGBoost
No	1	none	No	No	No	0.227	0.025	0.336	0.459
					Yes	0.047	0.025	0.342	0.364
				Yes	No	0.000	0.093	0.178	0.340
					Yes	0.000	0.071	0.116	0.309
		SMOTE	No	No	No	0.368	0.188	0.365	0.504
					Yes	0.377	0.000	0.328	0.407
				Yes	No	0.260	0.109	0.473	0.504
					Yes	0.294	0.172	0.306	0.484
		Bootstrapping	No	No	No	0.357	0.068	0.286	0.506
					Yes	0.376	0.088	0.308	0.456
				Yes	No	0.262	0.085	0.493	0.441
					Yes	0.296	0.069	0.314	0.490
	1,2	none	No	No	No	0.295	0.136	0.365	0.404
					Yes	0.219	0.071	0.229	0.330
				Yes	No	0.000	0.136	0.138	0.303
					Yes	0.000	0.182	0.048	0.247
			AdaBoost	No	No	0.081	0.000	0.288	-
					Yes	0.147	0.000	0.196	-
				Yes	No	0.024	0.000	0.138	-
					Yes	0.362	0.000	0.071	-
		SMOTE	No	No	No	0.332	0.176	0.286	0.414
					Yes	0.295	0.049	0.419	0.422
				Yes	No	0.338	0.154	0.345	0.496
					Yes	0.380	0.188	0.172	0.423
			AdaBoost	No	No	0.207	0.023	0.312	-
					Yes	0.231	0.000	0.456	-
				Yes	No	0.121	0.025	0.296	-
					Yes	0.132	0.108	0.172	-
		Bootstrapping	No	No	No	0.326	0.158	0.326	0.459
					Yes	0.289	0.216	0.167	0.497
				Yes	No	0.331	0.167	0.360	0.496
					Yes	0.389	0.047	0.172	0.496
			AdaBoost	No	No	0.239	0.088	0.321	-
					Yes	0.318	0.068	0.204	-
				Yes	No	0.317	0.112	0.357	-
					Yes	0.281	0.070	0.172	-
		ENN	No	No	No				
					Yes	0.190	0.095	0.152	
				Yes	No	0.000	0.072	0.116	0.306
					Yes	0.000	0.198	0.071	0.294
		SMOTE-ENN	No	No	No				
					Yes				
				Yes	No				
					Yes				

Table B.8: Results for computer trespass - part 2

Algorithm	Wordtype	N-gram	Weight	Filter	Resampling	**Cybercrime**	**Online threat**	**Online distribution of obscene imagery**	**Computer trespassing**
				No	No	0.2556	0.1622	0	0.1714
			Tf-idf	Yes	No	0.3718	0.2195	0.0833	0.1944
		1		Yes	SMOTE-ENN	0.2363	0.1556	0.1769	0.1746
Naive Bayes	Lemmas			No	SMOTE-ENN	0.3826	**0.2642**	0	0.1370
			Binary	No	No	0.1667	0.0448	0	0.0606
				Yes	No	0.3713	0.1875	0.0851	0.2222
Naive Bayes + AdaBoost	Lemmas	1	Tf-idf	Yes	No	0.3754	0.2346	0.125	0.2432
			Binary	Yes	No	0.3834	0.2013	0.0426	0.2667
				No	No	0.0866	0.0150	0	0.0313
			Tf-idf	Yes	No	0.2281	0	0.0455	0.1143
		1		Yes	SMOTE-ENN	0.2371	0.1882	0.1442	0.2095
Random Forests	Lemmas			No	SMOTE-ENN	0.2195	0.1786	0	0.15
			Binary	No	No	0.0759	0	0	0.0313
				Yes	No	0.1230	0.0145	0	0.1143
Random Forest + AdaBoost	Lemmas	1	Tf-idf	Yes	No	**0.4915**	0.1769	0	0.2254
			Binary	Yes	No	0.4255	0.2041	0	0.2
				No	No	0.2519	0.2105	0	0.1972
			Tf-idf	Yes	No	0.3962	0.2	0.1224	**0.2820**
		1		Yes	SMOTE-ENN	0.2359	0.1559	**0.1772**	0.1755
SVM	Lemmas			No	SMOTE-ENN	0.3946	0.2581	0	0.1867
			Binary	No	No	0.2114	0.0735	0	0.0308
				Yes	No	0.3630	0.1761	0.0833	0.2432
SVM + AdaBoost	Lemmas	1	Tf-idf	Yes	No	0.4207	0.0.1988	0.0833	0.2432
			Binary	Yes	No	0.3885	0.1975	0.0417	0.2703

Table B.9: Averaged F-scores of classifiers for preliminary results in the first phase (1896 instead of 3096 training instances)

Appendix C

Comparison graphs

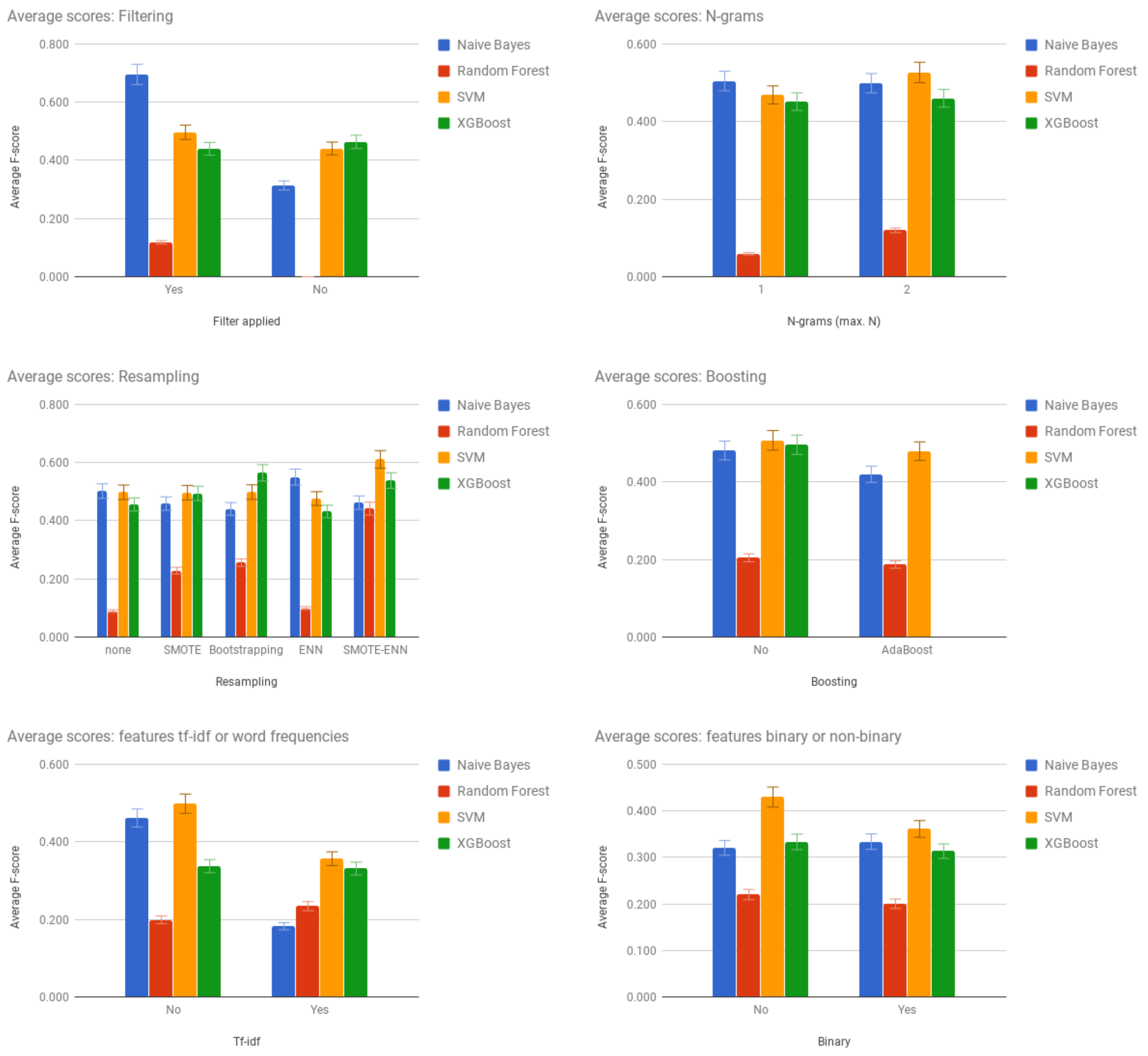

Figure C.1: Graphs comparing the scores of different models and model choices for the online threat category

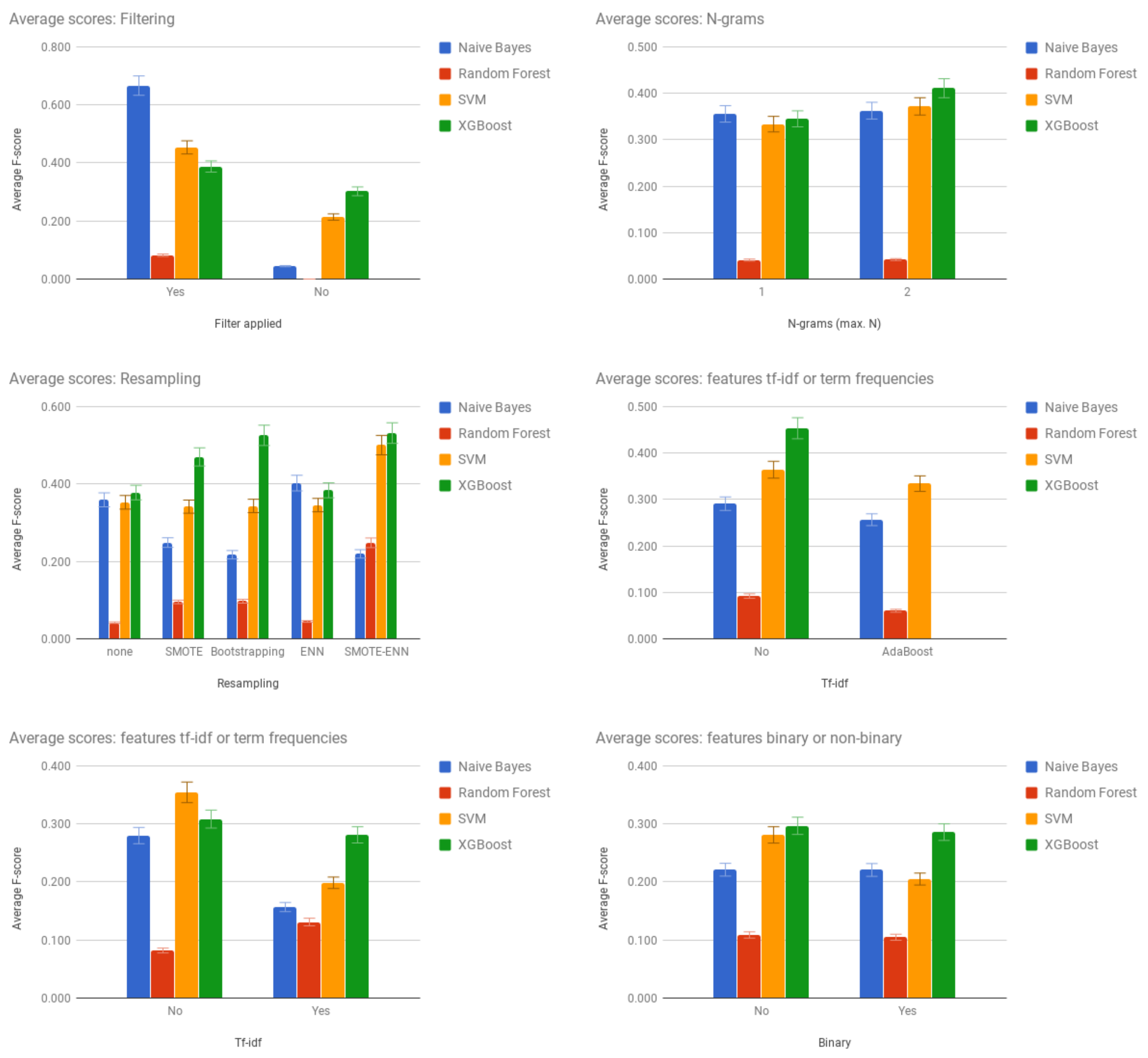

Figure C.2: Graphs comparing the scores of different models and model choices for the online distribution of sexually obscene imagery category

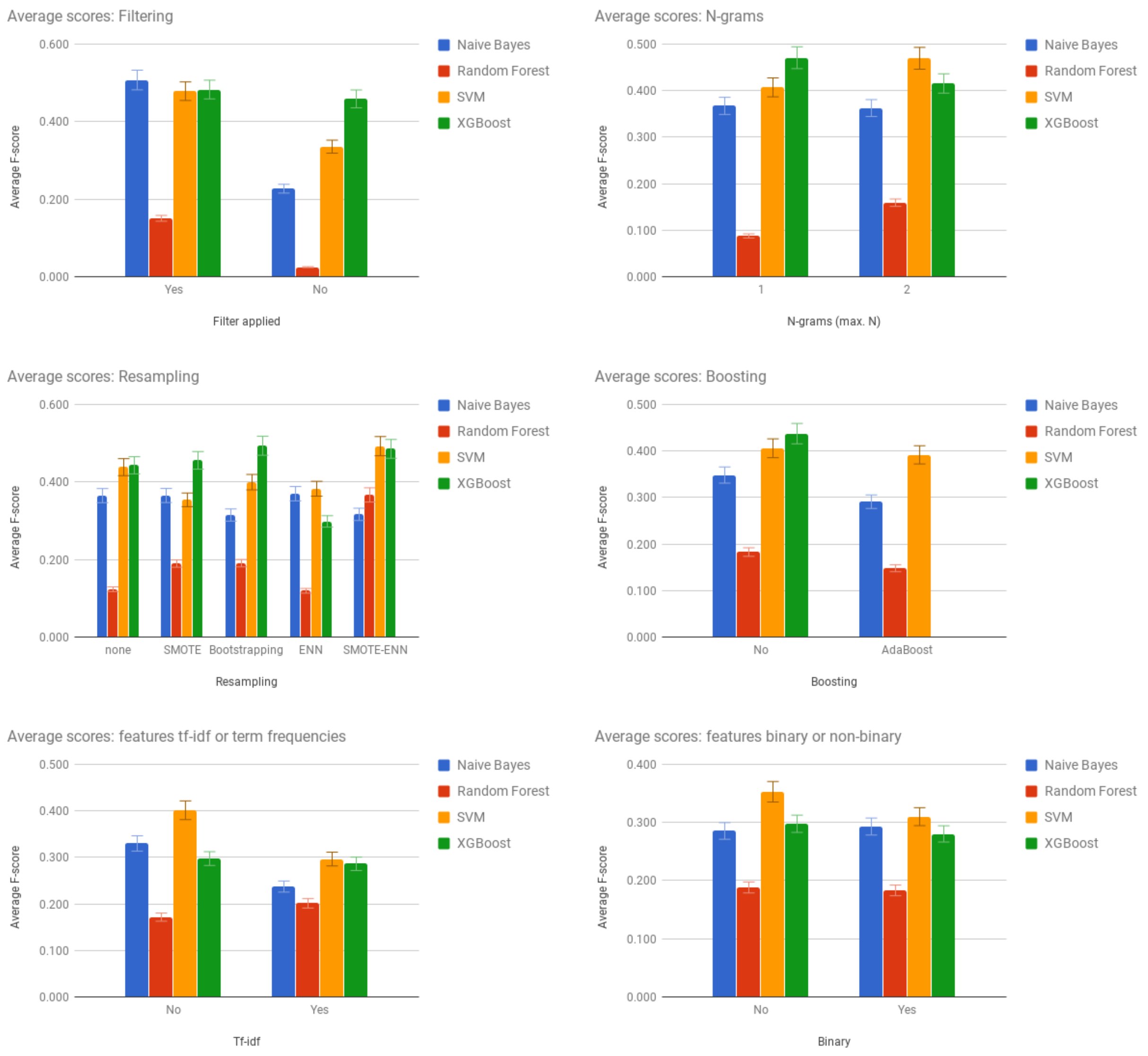

Figure C.3: Graphs comparing the scores of different models and model choices for the computer trespass category