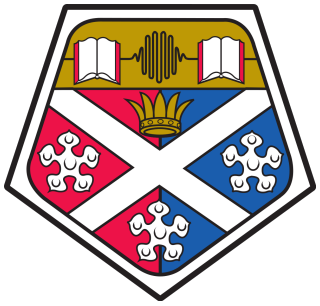


University of Strathclyde

Department of Computer and Information Sciences

Textual Analysis for Document Forensics

Alice Oberacker

This dissertation was submitted in part fulfilment of requirements for the degree of MSc Advanced Computer Science

August, 2017

Declaration

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

I declare that I have sought, and received, ethics approval via the Departmental Ethics Committee as appropriate to my research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive.
(please tick) Yes [ ] No [ ]

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is 19778.

I confirm that I wish this to be assessed as a Type
1 2 3 4 5 Dissertation (please circle)

Signature:

Date:

Abstract

In this dissertation text categorisation is applied to three datasets originating from the surface and dark web, two of which deal with extremism and drug related texts, whereas the third contains texts with four distinct contexts. Machine learning algorithms are applied to a numerical representation of the texts generated by a tool called Posit. This tool develops statistical data, such as average sentence length, and number of occurrences of parts-of-speech types and tokens. In addition to those values, bi-gram ratios are calculated from their frequency in the texts compared to their prevalence in the natural language.

The goal of this dissertation is to assess how effective the limited number of 30 features is for multiple classification algorithms. After conducting several experiments covering a wide range of settings, the research showed that on two of the three corpora the classifications based on the Posit and bi-gram features were evaluated to be very accurate. The third dataset demonstrated that classifying a text corpus with multiple contexts is difficult with the feature sets given.

This research showed that reducing a text corpus to its numerical information given by Posit is a powerful way of efficiently classifying large datasets.

Acknowledgements

I would first like to thank my dissertation advisor Dr. George Weir who is a lecturer at the University of Strathclyde. His continual guidance and professional suggestions helped me complete my Master's dissertation.

I would also like to thank Dr. Richard Frank who is Associate Director of the International CyberCrime Research Centre at Simon Fraser University in Canada. Dr. Frank supplied me with valuable data which made it possible to broaden my research perspective.

Finally, I want to express my gratitude to my parents who supported me through my years of studying and Dr. Philip Oberacker and Andrew Noble who constantly encouraged me through the process of researching and writing this thesis. This would not have been possible without them. Thank you.

Contents

1 Introduction **1**

2 Background and Literature Review **3**

2.1 Machine Learning 3

2.1.1 Data Scaling 5

2.1.2 Feature Selection 6

2.1.3 Validation 9

2.1.4 Evaluation 11

2.2 Text categorisation 14

2.3 Quantitative Text Analysis 17

3 Research Methods **21**

3.1 Approach 22

3.1.1 Hypotheses 23

3.1.2 Experimental Set-Up 24

3.2 Implementation 26

3.2.1 Data Preparation 26

3.2.2 Data Processing 34

4 Analysis **49**

4.1 Results 50

4.1.1 Hypothesis 1 51

4.1.2 Hypothesis 2 53

4.1.3 Hypothesis 3 55

4.1.4 Hypothesis 4 58

4.1.5 Hypothesis 5 63

4.2 Discussion 67

4.2.1 Hypothesis 1 67

4.2.2 Hypothesis 2 69

4.2.3 Hypothesis 3 70
4.2.4 Hypothesis 4 71
4.2.5 Hypothesis 5 72
4.2.6 Further Comments 73

5 Conclusion and Future Work **77**
5.1 Conclusion 77
5.2 Future Work 78

Appendix A Statistics **81**

Appendix B Implementation **85**
B.1 Posit Changes 85
B.2 Applying Posit 86
B.3 Applying Weka 109

Bibliography **127**

List of Figures

2.1 Over- ,under- and optimal fitting 4
2.2 Dimensionality Reduction with PCA 7

3.1 Distribution of number of words for all datasets. 36
3.2 Distribution of number of sentences for all datasets. 37
3.3 Distribution of average sentence length for all datasets. . . . 38
3.4 Correlation maps for all datasets. 39

List of Tables

2.1 Confusion Matrix 2×2 11
2.2 Aggregated Confusion Matrix over n classes 13

3.1 Posit numeric output. 27
3.2 Extract of Posit n-gram output. 28
3.3 Calculation of expected value. 29
3.4 Statistics of datasets compared 38

4.1 Attributes with their ID 49
4.2 Filter Methods. 50
4.3 Classification results for normalised and standardised DBpedia data for Posit and Posit + 2-gram attributes. 51
4.4 Classification results for normalised and standardised Extremism data for Posit and Posit + 2-gram attributes. . . . 52
4.5 Classification results for normalised and standardised Drug data for Posit and Posit + 2-gram attributes. 53
4.6 Classification results for normalised and standardised DBpedia data for Posit + low frequency 2-gram ratio and Posit + high frequency 2-gram attributes. 54
4.7 Classification results for normalised and standardised Extremism data for Posit + low frequency 2-gram ratio and Posit + high frequency 2-gram attributes. 55
4.8 Classification results for normalised and standardised Drug data for Posit + low frequency 2-gram ratio and Posit + high frequency 2-gram attributes. 56
4.9 Classification results for normalised and standardised DBpedia data only with the 2-gram features. 56
4.10 Classification results for normalised and standardised Extremism data only with the 2-gram features. 57

4.11 Classification results for normalised and standardised Drug data only with the 2-gram features. 58
4.12 Subset results for normalised and standardised DBpedia data 59
4.13 Classifier results for normalised and standardised DBpedia data; for normalised and standardised data S2 and S4 are equal. 60
4.14 Subset results for normalised and standardised Extremism data. 61
4.15 Classifier results for normalised and standardised Extremism data; S4 is equivalent to S3 for normalised data and S6 is equivalent to S5 for standardised data. 62
4.16 Subset results for normalised and standardised Drug data. . 63
4.17 Classifier results for normalised and standardised Drug data; S5 is equivalent to S6 for normalised and standardised data. 64
4.18 PCA classification results for normalised and standardised DBpedia data. 65
4.19 PCA classification results for normalised and standardised Extremism data. 66
4.20 PCA classification results for normalised and standardised Drug data. 67
4.21 Overall classification results for Hypothesis 1. 68
4.22 Overall classification results for Hypothesis 2. 69
4.23 Overall classification results for Hypothesis 3. 70
4.24 Overall classification results for Hypothesis 4. 71
4.25 Overall classification results for Hypothesis 5. 72

A.1 Drug Feature Statistics. 82
A.2 DBpedia Feature Statistics. 83
A.3 Extremism Feature Statistics. 84

Chapter 1

Introduction

Analysis of text data is a research topic going back more than 30 years. The task of classifying text documents to predefined categories has already been addressed in the 1980s using knowledge engineering techniques (Sebastiani, 2002; Jolliffe, 1972). This method requires the manual development of disjunctive normal forms to decide which category a certain text belongs to. Obviously, this approach can only be conducted with an expert of the domain and with the knowledge of an engineer who develops the algorithm. Even though the techniques showed positive results they are not quite applicable for the large amounts of data seen nowadays, as the development of such algorithms is expensive and inflexible towards expanding sets of documents. To give an example of how much data is circulating today one can take a look at the worldwide SMS traffic volume: between the years 2000 and 2012 the volume increased from 1.46 billion to 7.9 trillion (Informatica LLC, 2017). For this size of data more sophisticated methods must be used.

Machine learning, as it is known today, emerged in the early 1990s. One of the basic methods has not changed since: manually classified documents are used to train classifiers and to evaluate the results on a testing set.
In this dissertation three text corpora are going to be explored and analysed. Two of these datasets were created with web crawlers at the International CyberCrime Research Center (ICCRC) at Simon Fraser University in Burnaby, British Columbia. They cover themes such as extremism and drugs.
An example of a drug related text contained in the corpus is the following:

NL Growers − Coffee Shop grade Cannabis from the Netherlands − Weed, Hash, Marijuana, Cannabis for Bitcoins Products Login Register FAQs Coffee Shop grade Cannabis Finest organic cannabis grown by proffessional growers in the netherlands. We double seal all packages for odor less delivery. Shipping within 24 hours! Product Price Quantity 1g Original Haze 15 EUR = 0.027 Ƀ X 5g Original Haze 65 EUR = 0.117 Ƀ X 1g Bubblegum 10 EUR = 0.018 Ƀ X 5g Bubblegum 45 EUR = 0.081 Ƀ X 1g Jack Herer 14 EUR = 0.025 Ƀ X 5g Jack Herer 60 EUR = 0.108 Ƀ; X 1g Chronic 9 EUR = 0.016 Ƀ X 5g Chronic 40 EUR = 0.072 Ƀ X 1g Banana Kush 11 EUR = 0.020 Ƀ X 5g Banana Kush 45 EUR = 0.081 Ƀ X 1g Blue Cheese 9 EUR = 0.016 Ƀ X 5g Blue Cheese 40 EUR = 0.072 Ƀ X 1g Ice−O−Lator Hash, finest quality 35 EUR = 0.063 Ƀ X

The other text corpus contains 14 non-overlapping classes extracted from DBpedia 2014 (Lehmann et al., 2015), of these 14 categories four are going to be further appropriated for classification experiments.

In order to use machine learning algorithms to categorise texts one needs to find an appropriate representation. In most research the text corpora have an internal representation of large vector spaces, describing unique occurrences of words (Dumais et al., 1998; Sebastiani, 2005; Harish et al., 2012). A new interpretation of text documents was utilised by Weir (2007, 2009): The Posit tool. This tool extracts quantitative data from the text using parts-of-speech analysis and is going to be the main focus of this dissertation. The additional information extracted from these values, such as ratios between the number of word sequences appearing in the text and their frequency in the natural languages, operates as an enrichment to the data. This numeric data serves as the input for machine learning algorithms which try to determine a certain category the text belongs to.

The research aims at showing whether the quantitative approach is sufficient to categorise documents accurately. Furthermore, it will be analysed if there are features in the quantitative data more valuable than others. The importance of those word sequence features compared the Posit features will be evaluated. Additionally, the Posit features will be examined in order to make assumptions about the most important features, which could make it possible to further simplify the text representation. Different approaches to feature selection are going to be implemented and compared on the datasets.

Chapter 2

Background and Literature Review

This dissertation is based on a long-established research field which has grown to be of great importance when analysing large amounts of data. To set a common ground knowledge of this topic this chapter discusses several concepts of the field of machine learning. Furthermore, related research is going to be examined to place this thesis into context.
As the focus of this dissertation is on analysing texts in the context of a predefined set of topics, the following discussion will be centred around the problem of categorisation.

2.1 Machine Learning

Machine Learning focuses on bringing one part of a set of information in relation with the rest of the data. In our example this means to learn how a text relates to its theme, but it can also be used for other predictions, e.g. how customer behaviour is related to sales figures. The principle of machine learning lies in modelling of mathematical formulas that can represent the structure and information of the given data as closely as possible in order to apply the model to new data and predict the variable of interest.
There are two main types of machine learning algorithms: *supervised* and *unsupervised learning* (Sebastiani, 2002; Suthaharan, 2015). For the latter, the learning algorithm itself develops a set of categories suitable for the given data, whereas for supervised learning a pre-classified (labelled) dataset is necessary. Such data includes information about its categories

and the resulting model is later used to categorise data which has not been labelled beforehand. Having the focus on categorisation problems, supervised learning is also called classification and unsupervised learning is also known as clustering (Suthaharan, 2015).
For categorisation problems the model tries to find an optimal mapping between the information given by the data and a set of classes.
For a given set of data with k features the corresponding data domain D^k can be represented by a k-dimensional vector space. Therefore the mathematical model is a function f:

$$f : D^k \rightarrow \{1, ..., n\} \tag{2.1}$$

The target set of this function is a set of labels for n predefined categories (Suthaharan, 2015, chap. 1.3).

How well certain models perform on the individual datasets can be calculated with different measures such as accuracy, precision, sensitivity (Suthaharan, 2015, chap. 8.3), F_1 (Sebastiani, 2005) or the Receiver Operating Characteristic (ROC) (Aphinyanaphongs et al., 2014) (see Section 2.1.4). Common problems in machine learning are the so called *underfitting* and *overfitting* (Cios et al., 2007; Suthaharan, 2015): Figure 2.1 shows that a model that underfits the data does not reflect its variance accurately enough, this means that it has a considerably large error on the data the model was trained on. Overfitting on the other hand results in a too accurate classification of the data. The error is low on the learning data, but when applying the model to new data it will be unable to cope with the variance in the samples (Kohavi and John, 1997). The goal is to find an optimum between under- and overfitting, which balances missclassified samples in the training data against more accuracy when classifying new data.
There are several learning algorithms that are ap-

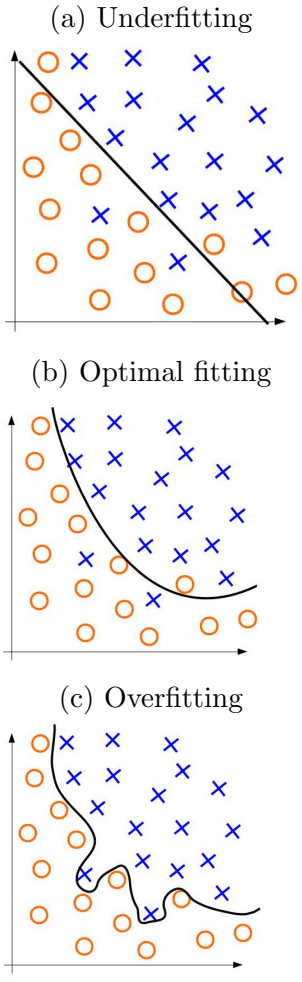


Figure 2.1: Over-,under- and optimal fitting

plicable for classification problems, such as Support Vector Machines (SVM), Decision Tree, Random Forest or Deep Learning (Suthaharan, 2015, chap. 1.3), some of them will be examined in the course of this dissertation. Furthermore, there are several ways to train, validate and test the resulting models, e.g. cross-validation. The concrete approaches taken and other options will be discussed in Section 2.1.3.
Another important research field is the feature selection. The goal of this is to find the subset of features from the data that leads to an optimal classification of the dataset and reduces the runtime of the classifier as e.g. 30 features require less computations as 300. The different approaches can be found in Section 2.1.2.

2.1.1 Data Scaling

Another crucial issue that arises with many datasets is the large variance among equally important features.
The features of dataset usually have different units and can therefore vary greatly in size. This disproportionality leads to the fact that learning algorithms cannot weight the importance of features correctly.
To circumvent this problem it is important to scale the data. Determining the optimal way for scaling the data cannot be decided uniformly, as it is dependent on the individual data. The most common techniques are:

$$\text{Normalising} \qquad \hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad \Rightarrow \hat{x} \in [0, 1] \tag{2.2}$$

$$\text{Standardising} \qquad \hat{x} = \frac{x - \mu}{\sigma} \qquad \Rightarrow \hat{\mu} = 0, \hat{\sigma} = 1 \tag{2.3}$$

$$\text{Centering} \qquad \hat{x} = x - \mu \qquad \Rightarrow \hat{\mu} = 0 \tag{2.4}$$

Where x denotes an element of a feature attribute, x_{min} and x_{max} are the minimal and maximal value of the feature respectively, μ is the mean and σ is the standard deviation of the original x. $\hat{\mu}$ and $\hat{\sigma}$ are mean and standard deviation of the newly calculated $\hat{x}$
Normalisation scales all values of a feature proportionally into the interval [0,1], whereas centering and standardising brings the mean to 0 and the

latter additionally makes the standard deviation of each feature 1.

2.1.2 Feature Selection

As we will see in Section 2.2 the most common approach to text classification takes a set of 10^4 to 10^7 features (Blum and Langley, 1997) which are used to train the classifier. However, research has shown that too many features are not only computationally expensive, but can also deteriorate the performance of the model (Langley and Iba, 1993; Blum and Langley, 1997; Kohavi and John, 1997; Janecek et al., 2008). Here three approaches for feature selection are going to be discussed: Filter, Wrapper and Principal Component Analysis.
For the former two methods search algorithms have to test possible feature subsets on how efficiently they predict the category, therefore these approaches belong to *supervised* methods. However, for k features this space has a size of 2^k which makes an exhaustive search too expensive. Heuristic search algorithms are therefore essential for large feature sets (Kohavi and John, 1997; Janecek et al., 2008).

Filter

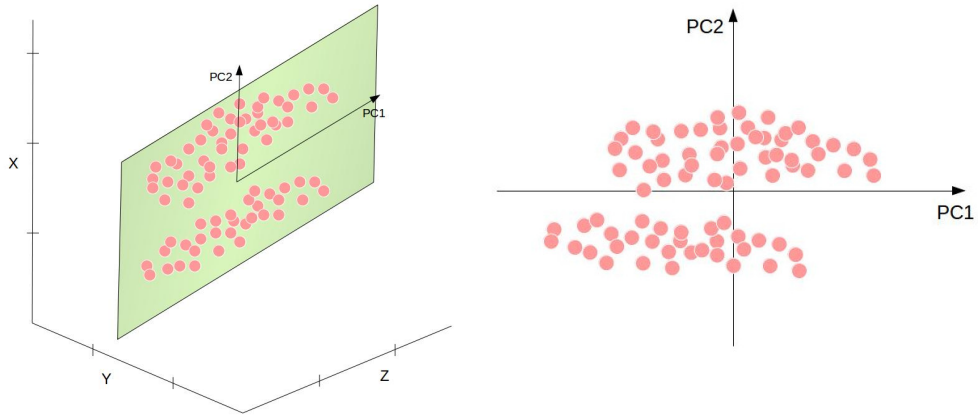
When using the filter method irrelevant features are filtered out by only relying on the training data. The importance of the individual features is evaluated using a shared measure (e.g. Information Gain) and therefore this approach is independent from any machine learning model (Blum and Langley, 1997). There are several combinations of search and evaluation measures possible. The ones being used in this dissertation will be further explained in Section 3.2.2.

Wrapper

The wrapper method is based on the learning algorithm. The space of feature subsets is evaluated by applying the learning algorithm and using the accuracy of the model based on the training data and the feature subset. Due to the computational expenses of this approach and depending on the size of the feature set it becomes even more important to utilise an efficient search algorithm.

Principal Component Analysis

Principal Component Analysis (PCA) belongs to the methods of dimensionality reduction. Its goal is to create a new feature space of linearly uncorrelated features, as they do not add any information (Jolliffe, 1972). In Figure 2.2 one can see a simplification of a PCA: a 3-dimensional dataset with features X, Y and Z is transformed into two dimensions: principal component 1 and 2 (PC1, PC2).



(a) Data Points in 3-dimensional Space (b) The same data points transformed to 2-dimensional space

Figure 2.2: Dimensionality Reduction with PCA

However, this approach does not extract the most valuable features (Guo et al., 2002), but transforms the whole feature space D^k into a less dimensional space D^m $(m < k)$. The new features $\hat{f}_i$ with $i = 1, ..m$ are defined as linear combinations of the original features f_j with $j = 1, ..k$.

$$\hat{f}_i = \alpha_i \cdot f_1 + \beta_i \cdot f_2 + ... + \omega_i \cdot f_k \tag{2.5}$$

To achieve this presentation of features a correlation matrix with dimensions $k \times k$ is created which describes the pairwise correlation between all features of the original set. From this matrix an ordered set of n $(n \geq m)$ eigenvalues with corresponding eigenvectors is determined, in which the eigenvectors are orthogonal to each other and define the new space. The eigenvalues and -vectors are ordered in such a way that the first feature has

the highest variance for the dataset and each following feature decreases in its variance. Therefore, choosing the first m features of this new space can be enough to cover a large amount of variance from the original space. The linear factors α_i to ω_i $(1 \leq i \leq m)$ for each new feature are the entries of the m resulting eigenvectors of the correlation matrix.(Suthaharan, 2015; Cios et al., 2007)
It is important to note that this approach is purely based on the variance of the dataset (Guo et al., 2002), it is therefore completely independent of the label (*unsupervised*) (Witten et al., 2011). Due to the importance of the variance of each feature, PCA can only be performed on scaled data. For normalised data the correlation matrix is used, for centred data the covariance matrix, and in case of normalisation by the standard deviation (standardising), both matrices are equivalent.

Blum and Langley (1997) report that the wrapper methods produce a more accurate subset of relevant features, as the filter method might use a different importance measure than the classification model. Hence, the result of the wrapper method agrees more with the classification model due to similar calculations. However, this also leads to the problem of overfitting (Kohavi and John, 1997), because the subset of features is marked as relevant based on the training data, but unseen data might relate well with other features which were judged to be irrelevant. Another disadvantage of the wrapper compared to the filter method are the high computational costs (Langley and Iba, 1993) as the learning algorithm has to be called for each feature subset considered. Guyon and Elisseeff (2003) claims, in a more recent publication, that coarse search methods not only improve the runtime of wrappers but also make the classifiers less prone to overfitting. Filter and wrapper methods aim on removing superfluous features, however, if too many features are removed, because the subset is due to be really small, important information can get lost. In this case, PCA can perform better, as it keeps all features of the original set and therefore does not lose much information by weighting the features differently. Nonetheless, reducing the dimensionality of the data makes it harder or even impossible to assess the importance of the original features (Janecek et al., 2008).
This dissertation will consider both, filtering and PCA, as feature selection methods in order to compare their efficiency on the data. The wrapper method is not going to be part of this due to its computational demands.

2.1.3 Validation

Having seen that supervised learning algorithms use a subset of the data to evolve a mathematical model this section will talk about the validation process of the model and different ratios for splitting the dataset. There are other techniques which train the model on the whole dataset and use a subset of the already seen data to measure how well the model performs (Suthaharan, 2015). Here only the testing on unseen data will be considered, because the datasets are large enough to exclude a subset for testing and because evaluation on already seen data does not reflect the same capabilities as the performance of never seen samples does.

Train – Test

The most basic approach to validating a learning model is to train it on a subset of data and to test it on the leftover samples. Splitting the data into two disjunctive sets only raises one question: What ratio between testing and training data should be chosen?
According to Suthaharan (2015, chap. 8) 80:20 or 70:30 are acceptable choices. Witten et al. (2011, chap. 5.3) considers 75:25 as a common choice and points out that particularly for classification problems it is important that every category is represented by a sufficiently large amount of data in both subsets. The process of equally distributing the categories over the subsets is called *stratification*. Otherwise, the model might not have enough or any information about a category and can therefore not make any decision when coming across that category in the testing set (Witten et al., 2011).

Train – Validate – Test

Another popular approach uses three disjunctive sets of the data, split by a ratio of 60:20:20. Similarly to the previous one, the training data is used to design the model and the testing set to judge its accuracy. However, validating the model after training makes it possible to go back and train a different model. Suthaharan (2015, chap. 8) explained the process as follows: Imagine training two different learning algorithms M_1 and M_2 with errors E_1 and E_2. The errors show how well the model classifies the training data. Now both models are applied to the validation set with resulting

accuracies A_1 and A_2. Let's say $E_1 < E_2$: if $A_1 > A_2$ the validation process showed that M_1 performed best on training and validation set. Otherwise, if $A_1 < A_2$ then M_2 performed better on the validation set than M_1 which contradicts the error values of the training phase. Therefore, new models can be trained until the error of the training set agrees with the accuracy of the validation phase. If this is achieved the final model can be tested on the leftover data.

Cross – Validation

There are several versions of cross-validation including *k-fold Cross-Validation*, *Leave-One-Out* and *Leave-p-Out*.
The Leave-p-Out cross-validation takes the set of instances and creates pairs of sets of which one holds p-many and the other one holds the rest of the samples. Therefore, the set pairs are disjunctive. When creating all possible combinations of sets for n instances one has $\binom{n}{k}$ many pairs. For 5 samples $\nu_1 ... \nu_5$ and $p = 2$, for example, the cross-validation is calculated on $\binom{n}{k} = 10$ combinations.
The following set of pairs are created:

$$\begin{array}{lll} \{\nu_3,\nu_4,\nu_5\}\{\nu_1,\nu_2\} & \{\nu_2,\nu_4,\nu_5\}\{\nu_1,\nu_3\} & \{\nu_2,\nu_3,\nu_5\}\{\nu_1,\nu_4\} \\ \{\nu_2,\nu_3,\nu_4\}\{\nu_1,\nu_5\} & \{\nu_1,\nu_4,\nu_5\}\{\nu_2,\nu_3\} & \{\nu_1,\nu_3,\nu_5\}\{\nu_2,\nu_4\} \\ \{\nu_1,\nu_3,\nu_4\}\{\nu_2,\nu_5\} & \{\nu_1,\nu_2,\nu_5\}\{\nu_3,\nu_4\} & \{\nu_1,\nu_2,\nu_4\}\{\nu_3,\nu_5\} \\ & \{\nu_1,\nu_2,\nu_3\}\{\nu_4,\nu_5\} & \end{array}$$

This approach is usually too computationally expensive, the Leave-One-Out validation process, however, is less expensive as it takes the same approach with $k = 1$ and has therefore $\binom{n}{1} = n$ many combinations. Besides the costs of executing this validation method it has the advantage that no random sampling is involved. Each instance of the data is used once for testing and in all other cases as training. Therefore the validation result does not change when repeating the process (Witten et al., 2011, chap. 5.4).
The k-fold Cross-Validation is less expensive and seems to be the most popular one (Forman, 2003; Huang and Wang, 2006; Lewis et al., 2004; Kohavi and John, 1997). Let's consider a stratified 10-fold cross-validation: The whole dataset is split into ten stratified subsets. In every fold one of the subsets is used as a testing set and the rest as training data. Therefore there are ten folds and each subset is used once as testing data and every

other time as training data.
The general approach stays the same as before: training data is used to create a model and testing data to evaluate its accuracy. Hence, there are ten resulting accuracies which are averaged to create an overall accuracy of the model.
According to Witten et al. (2011, chap. 5.3) there have been extensive studies on what validation methods show the best results and the 10-fold cross-validation became the standard method as it has shown to be the most efficient validation process on various datasets and with different learning algorithms.

2.1.4 Evaluation

In order to judge how well the trained classifiers perform on new data we need to introduce evaluation metrics. As only classification models are discussed here numerical performance measures such as root mean-square or mean absolute error etc. are going to be neglected.
Furthermore, there are two cases that have to be considered separately: the binary and the multi-class categorisation. Different evaluation measures have to be used for those cases.

Binary Classification

This type of classification deals with two distinct categories: one positive and one negative towards some objective.
Using a *confusion matrix* several performance measures can be calculated. The confusion matrix as seen in Figure 2.1 is a 2×2 matrix where the rows define the actual classes and the columns the predicted classes. One class is defined to be the positive class (yes) and the other one to be the negative class (no).

		Predicted Class	
		yes	no
Class	yes	true positive (TP)	false negative (FN)
	no	false positive (FP)	true negative (TN)

Table 2.1: Confusion Matrix 2×2

Correctly classified instances are *true positives*, if they predict the positive

class correctly, or *true negatives*, if predicting the negative class correctly. A *false negative* denotes an instance that is predicted to be negative but is actually positive and a *false positive* instance was predicted to be positive even though it is negative.
With this information several confusion-based performance measures can be defined. The accuracy is defined by the proportionality between the correctly classified samples and the overall number of classifications (Witten et al., 2011).

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6}$$

However, one needs to be careful with this type of metric. When considering models that always predict the largest class of the training data, the accuracy is high, even though the model has no predictive power. If the positive class is the largest one then $TP < FP$, otherwise $TN < FN$. This is described as the Accuracy Paradox (Zhu, 2007).
The precision returns the ratio of how many positive predicted samples are actually positive.

$$precision = \frac{TP}{TP + FP} \tag{2.7}$$

Contrasting that with the recall (or sensitivity (Suthaharan, 2015)) one can calculate the ratio of how many samples of the positive class were predicted correctly.

$$recall = \frac{TP}{TP + FN} \tag{2.8}$$

These two measures are less sensitive to skewed classifications, however, precision should be preferred in the case of expensive false positive results, whereas recall performs better if false negatives are expensive. In the case of perfect classification (FN=0=FP) precision and recall are 1.
Specificity measures how many times the negative class is correctly classified as negative. Cios et al. (2007) phrased it as 'the ability of a test to be negative when the disease is not present'.

$$specificity = \frac{TN}{TN + FP} \tag{2.9}$$

The F-measure, also known as F_1, is calculated as the harmonic mean of precision and recall (Cios et al., 2007).

$$F - measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.10}$$

One can conclude that a combination of precision and recall is a stable measure when dealing with skewed classes as the accuracy paradox can be avoided. To present an overall measure of precision and recall the F-measure can be applied.

Multi-Class Classification

Having seen several performance measures for binary categorisation, all based on the confusion matrix, we can now proceed to the classification of multiple categories.
Fortunately, the previous techniques can be adopted. When dealing with n categories, the confusion matrix has the same structure. The rows show all categories and the columns the predicted categories, therefore the matrix has dimensionality of $n \times n$. Another way of representing the confusion

		Predicted Class	
		yes	no
Class	yes	$\mathrm{TP} = \sum_{i=1}^{n} \mathrm{TP}_i$	$\mathrm{FN} = \sum_{i=1}^{n} \mathrm{FN}_i$
	no	$\mathrm{FP} = \sum_{i=1}^{n} \mathrm{FP}_i$	$\mathrm{TN} = \sum_{i=1}^{n} \mathrm{TN}_i$

Table 2.2: Aggregated Confusion Matrix over n classes

matrix for the multidimensional case is to aggregate over all categories (Sebastiani, 2002) as shown in Table 2.2. Exemplary shown on precision and recall, it is now possible to calculate them by averaging over the whole set of categories, called *microaveraging* (Sebastiani, 2002).

$$precision^{\mu} = \frac{\sum_{i=1}^{n} \mathrm{TP}_i}{\sum_{i=1}^{n} \mathrm{TP}_i + \mathrm{FP}_i} \tag{2.11}$$

$$recall^{\mu} = \frac{\sum_{i=1}^{n} \mathrm{TP}_i}{\sum_{i=1}^{n} \mathrm{TP}_i + \mathrm{FN}_i} \tag{2.12}$$

Another way of calculating the average over TP, FP and FN is *macroaveraging* (Sebastiani, 2002). Precision and recall are calculated individually for each category and are then summed up and divided by the number of categories.

$$precision^M = \frac{\sum_{i=1}^{n} \text{precision}_i}{n} \tag{2.13}$$

$$recall^M = \frac{\sum_{i=1}^{n} \text{recall}_i}{n} \tag{2.14}$$

The μ indicates microaveraging and M macroaveraging.
Microaveraging is prone to putting more weight on categories with many positive samples, however, which type of averaging performs the best can not be decided generally, but is dependent on the dataset (Sebastiani, 2002). Obviously, the same calculations can be applied to accuracy, specificity and the F-measure.

2.2 Text categorisation

As previously mentioned, Sebastiani (2005, 2002) defines two main classes of text categorisation: text clustering and text classification. The first one deals with finding a structure of groups among the dataset, whereas the latter is given a set of groups and each text needs to be assigned to its most suitable one. Moreover, the task of text classification is subjective in a way that human and machine might disagree on the classification of the data.
Text classification can be single-labelled meaning every document is assigned a single category, or multi-labelled in which case a document can be assigned to several possible categories. This method has the advantage of giving the user the possibility of a final decision to their own subjective opinion as several texts can be closely related to multiple categories.
Several applications such as spam filtering, webpage classification, authorship attribution or genre classification can be decided with text classifications. Among the various machine learning algorithms that have been used to build classifiers, Sebastiani (2005) claims the ones that proved most successful in recent years are support vector machines (SVM) and boosting. SVM is a type of classification model, boosting, however, combines the decisions of a group of classifiers in order to achieve a better overall classi-

fication (Sebastiani, 2005). Lewis et al. (2004) agrees on the effectiveness of SVM, but also points out that this approach might find a suboptimal decision threshold for categories with low occurrences.
However, it remains a challenge to achieve high accuracy for all possible contexts at once, as no algorithm is most effective on all applications (Harish et al., 2012; Aphinyanaphongs et al., 2014). Moreover, the labelling of the documents defines a bottleneck for every supervised classification method as it has to be done manually.

To solve this problem Ha-Thuc and Renders (2011) developed a system to hierarchically classify unlabelled data. As already mentioned, classifying data manually is extremely expensive and slows the classification process down. Additionally, it grows to be an inefficient approach as with larger datasets the number of categories can exceed to thousands, of which each needs to be represented by a sufficient amount of labelled documents. The system solves this issue by using ontological knowledge and by searching 'pseudo-relevant documents on the Web' (Ha-Thuc and Renders, 2011). With the ontology it is possible to create a hierarchical model including the context of ancestors among different classes.
Depending on the dataset this might be a necessary approach on solving the bottleneck problem. However, such a concept exceeds the scope of this Master's thesis and therefore a domain with a small number of categories will be chosen for research.

Lewis et al. (2004) compared the accuracy of SVM, k-Nearest Neighbours (k-NN) and Rocchio-Style Prototype Classifier with each other on the Reuters Corpus Vol. 1. Two variants of SVM were used. The first one was trained for each category by using the default settings and the latter tried to find optimal settings to improve results for unbalanced classes for each category and was trained using a leave-one-out cross validation. Results show that the first SVM classifications achieve the best F_1 values, followed closely by the second SVM approach. k-NN and Rocchio-Style did not achieve as good results, which underlines the statement made by Sebastiani (2005).
Another study by Harish et al. (2012) compared results of k-NN, Rocchio-Style and Linear Least Square Fit (LLSF) with each other. Throughout the experiment k-NN achieved the best classification results, with Rocchio and LLSF showing reasonable efficiency. Harish et al. (2012), however, state

that SVM methods can be used to improve upon the k-NN results. The k-Nearest Neighbour method is a lazy learning method, because few calculations are done during the training phase. During the classification the distances to all training samples have to be calculated to find the k nearest samples, which makes it a lazy learning method and therefore more sensitive to noisy data as it only considers a few samples to make a decision (Witten et al., 2011, chap. 4).

For classifying text corpora one has to develop an internal representation for the learning algorithms. The most common approach represents each text as a vector in which every position displays the existence of a word (set of words). Similar techniques do not only acknowledge the existence but also the frequency of words (bag of words) (Forman, 2003; Sebastiani, 2002). The representation usually has a large number of features due to the number of unique words in the document. Therefore it is appropriate to remove irrelevant features to optimise the prediction (Aphinyanaphongs et al., 2014), as was discussed in Section 2.1.2. However, it needs to be shown if feature selection plays an important role when using the Posit toolset, as the number of features that can be extracted from the computed quantitative data do not expand the runtime of the learning algorithms drastically. As it is suggested in multiple papers (Aphinyanaphongs et al., 2014; Forman, 2003; Joachims, 1998) feature selection can improve the performance of classifiers. Therefore the dissertation will further analyse the importance of certain features of the Posit tool to discuss their influence on the classification.
Forman (2003), for example, points out that words with low frequencies can be neglected as well as so called stopwords, such as 'a' and 'or'. However, for every approach one needs to bear in mind the possibly varying size of documents as the occurrences need to be normalised over the size of text.
Aphinyanaphongs et al. (2014) suggest that the most suitable classification performance metrics is the receiver operating characteristic (ROC), which plots sensitivity against 1−specificity. The area under curve (AUC) can then be used to differentiate between perfect classification (AUC=1), classification by chance (AUC=0.5) and inverse classification (AUC=0). The advantage of this metric is its insensitivity to unbalanced categories.

In the field of computational linguistics *n-grams* are defined as a sequence of characters or words of length n. The Posit tool makes it possible to ex-

tract word grams of length 2, 3 and 4 including their frequency. Statistical features about word n-grams have been appropriated for text classification by Peng and Schuurmans (2003). The n-gram language model is handled in a similar way to a Naïve Bayes model. Each category is trained with a language model and every document can be evaluated on each of those models to decide to which it agrees the most. In this experimental paper it was shown that statistical data of n-grams can be used for a chain augmented Naïve Bayes classifier. An optimal size for n-grams can be found to improve the classification of documents. This underlines the importance of the Posit tool to quantify data about n-grams. It seems to be a source for improving upon classification accuracy.

2.3 Quantitative Text Analysis

The most popular approach to text classification is representing each text as a vector of word occurrences (set or bag of words) (Harish et al., 2012; Forman, 2003; Cios et al., 2007; Sebastiani, 2002). One way of modelling such a vector is denoting the occurrence of a word by setting the position to 1 and otherwise to 0. There are other models which also include the frequency of which a word appears in the text which can be of great importance to the classification.
However, these approaches require a lot of computational time and optimisation, for example using feature selection (see Section 2.1.2).

Another approach, which enriches the representation of texts for machine learning models is going to be discussed in this section and will be of great importance for the rest of this dissertation. Instead of representing a text with its words, the following tool calculates quantitative and statistical values for a text which are suggested to be valuable for classification models.

The Posit Text Profiling Toolset (Weir, 2007, 2009) is based on Unix scripting and proved to be applicable to large text files, because interim results are stored in temporary files, which makes the process more memory efficient.
The features of the Posit tool can be split into three modules:

– The parts-of-speech (POS) Profiler develops statistics about the POS characteristic in a given document. The parts-of-speech are further

distinguished into types and tokens, where types define the unique occurrences of POS features.
These statistics include frequencies for parts-of-speech tokens and types, type-token ratios, number of sentences and average sentence and word length. Further information is gathered by aggregating the data into subgroups, for example common nouns and proper nouns or for the different verb tenses.
The most detailed statistic is the total frequency of each POS token or type.

- The Vocabulary Profiler can extract the least common words in a text compared to a reference frequency list. Using a statistical significance measure on the frequency of words compared to a reference frequency, this can then produce keywords for the text. The goal of this procedure is to give the author feedback of the chosen vocabulary. Moreover, the tool is able to determine n-gram frequencies. This enables the user to compare word and n-gram frequencies of a text with a reference frequency list.

- The Readability Profiler, which is under development, will use the results of the previous modules and a collocation analysis to detect the readability of the document. As for the Vocabulary Profiler, the collocation frequency can be compared to an average collocation frequency measure to rate readability among texts.

Additionally to these modules the tool offers the possibility to use POS tagging on the documents. In order to understand the context for specific keywords one can utilise the concordance option of the tool. This makes it possible to find word spans around the desired keyword.
One can conclude that the Posit tool offers a thorough quantitative analysis of an arbitrarily large text corpus with highly customisable features. The calculated variables can be further employed for research purposes on textual classification.

Weir et al. (2016) applied the Posit tool on the data retrieved by the Terrorism and Extremism Network Extractor (TENE) webcrawler. The vast amount of online data makes automatic classification absolutely essential. TENE was developed by the International CyberCrime Research Center (ICCRC) at Simon Fraser University in Burnaby, British Columbia.

The basis of the classification process is the manual classification of the text data into the categories 'pro-extremist', 'neutral' and 'anti-extremist'. The conducted research was twofold: The group at ICCRC based the classification on sentiment analysis. In order to apply this analysis POS tagging was used to extract key nouns and their frequency distribution. Then SentiStregth was applied, which is a tool for analysing the sentiment of words by assigning numbers to words according to their positive or negative connotation. As a result, the extracted keywords were labelled with positive or negative values.

The researchers at the Computer and Information Science department at the University of Strathclyde in Glasgow (UST) analysed the quantitative syntactic features derived by the previously discussed Posit toolset in order to enrich the information given by the text corpus.

The sentiment analysis approach achieved a correct classification of 80.51% with a standard J48 decision-tree, whereas the Posit approach classified 91.4% of the webpages correctly with the same classifier. The best result of 95.3% correctly classified texts was accomplished with a Random Forest classification on basis of the Posit data.

These results clearly show that the quantitative data performs better classifications, even though sentiment analysis might be considered as a more suitable tool due to the emotional subject of the webpages. One reason might be the insufficient data that was extracted for sentiment analysis, as only nouns were considered whereas the Posit tool exploited a much richer set of features.

For further research it would be interesting if an in-depth sentiment analysis of several parts-of-speech can improve the classification results. Considering the Posit method it could be informative to analyse which features extracted by Posit are the most important to classify a dataset as such.

Previous research conducted by Weir and Anagnostou (2007) applied the same analysis of parts-of-speech as Posit to categorise newspaper articles. The most frequently used words of three categories were considered to make statements about the articles. The research detected a classification problem within the quantitative approach. When comparing the distribution of parts-of-speech with the British National Corpus divergences were found. It needs to be mentioned that it is not to expect that a general set of texts has similarities to a specific set of documents as the newspapers. However,

this comparison showed variance in distribution of nouns, 'non-specified' parts-of-speech and determiners. A re-examination revealed that the POS tagging tool mistyped terms as too many words were wrongly classified as nouns, not only due to weaknesses in the tool but also due to typographical errors.

Undoubtedly, these issues caused by POS tagging influence the further research based on the word frequencies. The quantitative data produced must therefore be examined carefully in order to perform representative research on text classification.

Chapter 3

Research Methods

The previous chapter dealt with text categorisation in general by discussing how texts are appropriated in order to feed them into machine learning algorithms. A more specific approach dealing with quantitative data extracted from text corpora has been examined as well. The research suggested that numeric data about the linguistics of a text has a large impact on classification efficiency of machine learning algorithms. The advantage of a quantitative approach, opposed to a vector representation of the existence of words in the text (bag of words), is that the number of features is much lower. Instead of dealing with millions of features (Blum and Langley, 1997), the Posit tool extracts 27 distinct values. Furthermore, Posit allows collection of frequency data about n-grams, which will be important for this research. In combination with reference data concerning the natural English language, it is possible to extract frequency ratios which could further enrich the feature set.
Some important steps of the machine learning process were reviewed in Section 2.1, including feature selection. This comes together now with the need to analyse the importance of n-gram and other features of the Posit output.

After settling on the aim of this research in the first section of this chapter, the latter part will deal with the concrete preparation of the data and the implementation of the experiments is going to be examined.

3.1 Approach

The Posit tool has served to create data from text for only one corpus before: extremism data extracted from the World Wide Web (Weir et al., 2016). In this research the effectiveness of the Posit features are going to be further evaluated on the basis of three datasets with varying topics. In the following 'Posit features' will refer to the summary data generated by Posit's core functionality.
The aim of the experiments circles around the importance of feature subsets. There are several constellations that can be evaluated: it will be of interest to analyse if only a subset of the Posit features is sufficient to create a well working machine learning model. This raises the question, if a subset of most important features is dependent on the dataset or topic of data or if that subset proves to be important among several corpora.
Moreover, using numeric values based on n-grams has, to our knowledge, not previously been considered by other authors. The frequency data of n-grams can be extracted using the Posit tool. Analysis should be performed to see if one can take advantage of such values and if so, how much more precision of the classification model can be achieved. It needs to be considered if n-gram features work well on their own or if they, combined with a subset of Posit features, improve the machine learning approach.
Another aim of this research is to find out if n-gram features influence the importance of the Posit features. In combination with n-gram features, previously less important Posit features could contribute to a better classification. Again, these facets need to be separately analysed on the different datasets. Only then can one draw conclusions about the likeliness that important features are independent on the dataset or vary among them.

In order to evaluate how well each set of features performs several machine learning models are going to be assessed. Depending on the used models, option tuning can be performed to achieve better results.

These thoughts are going to be further elaborated and summarised in the next section by defining hypotheses. It is important to maintain a consistent experimental set-up to draw valid conclusions, which will be discussed as well in this chapter.

3.1.1 Hypotheses

The foundation of an experimental research is to construct hypotheses on the grounds of research questions. These will then be evaluated to prove or disprove their statement. In the following section several theories concerning feature subsets will be formulated.

n-gram Features

The first aspect that is going to be appointed is the contribution of n-gram ratios to the classification model. The frequency of n-grams could contain valuable information as sequences of words preserve more context than individual words. n-grams with lower frequency than the average n-gram in the natural language could therefore be of greater significance as they point out distinct properties of a text.
Another theory involving n-grams is that there are vital Posit features without which the classification model does not work well. This implies that using only the features created from n-gram frequencies will not lead to accurate predictions.

Hypotheses:

1. *n-gram features improve the classification accuracy of machine learning algorithms independently of the data domain.*

2. *A feature created from low-frequency n-grams is more important than a feature of high-frequency n-grams.*

3. *n-gram features alone are not sufficient for classification models to perform well.*

Subset of Features

It is likely that a subset of the Posit and n-gram features serves as a better basis for classification models as some features are correlated, e.g. the average sentence length is calculated as

$$\text{average sentence length} = \frac{\text{number of words}}{\text{number of sentences}}.$$

By selecting uncorrelated features it should therefore be possible to optimise the performance of the classifier (Langley and Iba, 1993; Blum and

Langley, 1997; Kohavi and John, 1997; Janecek et al., 2008). To explore this feature selection can be applied to the full set of Posit and n-gram features.
Another aspect of feature selection that can be considered here is Principal Component Analysis, which keeps all features, but creates new ones based on linear combinations of the original features. Within each new feature every old one is weighted by a factor. PCA creates features that are uncorrelated and therefore it is expected that this method would improve the predictive power of the classifier.

4. *Using a subset of Posit and n-gram features leads to better classification results.*

5. *Using Principal Component Analysis improves the effectiveness of the features and leads to better classification results.*

3.1.2 Experimental Set-Up

Having defined the hypotheses that will be targeted it is now time to design the experiments. Bearing in mind that only the variables of interest can be altered and the rest of the environment must be stable, several steps of the machine learning process are consistent among all experiments.

Data Preparation

Undoubtedly, the data must be preprocessed before any of the tests can be conducted. Starting with the raw text data, firstly the Posit data has to be generated including the n-gram frequencies. Once the data has been converted into its numerical representation further preprocessing steps must be taken. This includes cleaning out noise and dealing with missing values as well as scaling the data (see Section 2.1.1). Furthermore, to have precisely the same data configuration one should conduct the studies on the same training set. Therefore, the part of data splitting (see Section 2.1.3) has to be done beforehand as well. Each data corpus will be split into 70% training data and 30% testing data after scaling it using normalisation and standardisation. As the sampling of the data is random it should not influence the outcome of the experiments, but if the machine learning model is unstable the sampling can have a great impact on the experiments. A more accurate approach would be to apply a k-fold cross validation. Due

to the computational costs and the number of experiments to be conducted it is more feasible to perform a 70:30 sampling.

Experiments

For hypotheses 1 and 3 a selection of features can be done manually. Case 1 involves two settings: first the classification takes place based solely on the Posit features and then including the n-gram ratios. The measures of the used learning algorithm can then be compared for the different datasets.
Case 3 involves selecting only the n-gram features for classification. The results can be compared with the evaluation of case 1 or assessed on their own.

There are several ways to conduct experiments for hypothesis 2. The first possibility is to manually select the Posit features and either one of the n-gram features. Then the evaluation measure can be used as an indicator if the low-frequency or the high-frequency feature was more valuable to the classification.
Another possibility would be to use the full set of Posit and n-gram features and then to apply the filter or wrapper methods discussed in Section 2.1.2. Using those methods the features can either be ranked by importance or a set of the most important ones will be returned. In the first case it is clear how to assess the values of both n-gram features. In the latter case a difference in importance can only be determined if either of the features is not contained in the returned subset. If both or neither are included, then no statement towards the hypotheses can be made. Another important point to be mentioned is that the filter methods use different measures to asses the benefit of including certain features than the final classification algorithm might use. Therefore, even if a feature is predicted to be of greater significance than others, the machine learning algorithm might not be able to exploit the full range of profit. Consequently, it is necessary to apply a classification model to evaluate the prediction performance of the individual feature subsets.
After these considerations the first approach seems to offer clearer indications on how important each feature is, thus the second approach will not be executed.
These thoughts about feature selection directly lead to examining hypothesis 4 which suggests that features can be discarded from the classification

model to improve its accuracy. As discussed in Section 2.1.2 there are several methods to choose from in order to search and evaluate possible subsets of features. The resulting sets can then be further evaluated against each other by applying learning algorithms and the achieved accuracies can also be compared against the results of hypothesis 1.
The same holds for hypothesis 5. Principal Component Analysis can be applied with different settings of covered variance. The resulting accuracy of the PCA can be compared with the results of the feature selection in hypotheses 4 and the accuracies achieved in hypothesis 1 when all features were applied.

3.2 Implementation

After having defined the concepts of research to be conducted, this section will deal with the concrete implementation.
All datasets were received as raw text files which had to be transformed into numeric data. The first part of this section will deal with the preparation of the data, followed by the individual machine learning procedures such as getting to know the datasets, cleaning them, selecting features and finally applying the classification models.

3.2.1 Data Preparation

In this case, having three different datasets available means being able to to cope with three different inputs.
The DBpedia data was stored in two separate text files, one for training with 560 000 texts and one for testing with 70 000 texts. Each file contained one text per line.
The extremism dataset was split into three folders representing its categories positive, negative and neutral with an average of 6 400 texts in each set. Every text was stored in its own file with additional information in the first line.
The drug dataset was available in a similar configuration, the only difference being that the files were not stored per category but per crawl and within that per domain. Each file only contained the text with no other data. The full dataset included 1 245 410 files many of which had to be discarded from the research for reasons discussed in Section 3.2.2.

In order to receive fast results it was important to split the large amount of texts across several machines. For the first dataset the approach to perform the transformation on subsets of lines was evident. A slightly altered approach was used for the other corpora. The file names including the folder names were stored to index files – each line containing the path to one text file. Then the process of extracting the numeric data was applied on subsets of lines in order to execute the transformation in parallel. Each execution created one file including the numeric data of all text files processed in that run. Finally, the resulting files were combined to one for each dataset.

Posit

The Posit tool consists of several scripts computing the numeric data. These numeric values are threefold in their meaning: first, statistics about word count, number of characters and sentences are computed; second of all, number of token types are aggregated; finally, the part-of-speech (POS) types are returned. A full list of all values can be found in Table 3.1, as one can see the word types for token and part-of-speech are equal.
The Posit tool also has the capability of extracting 2-, 3- and 4-grams and their frequency in the text. An example of the bi-gram output can be seen in Table 3.2. As this dissertation is of experimental nature only bi-grams will be considered. The script executing these computations was modified in such a way that it only created output for bi-grams. The core logic of

Statistics	Token/POS Types
Total words (tokens)	noun
Total unique words (types)	verb
Type/Token Ratio	adjective
Number of sentences	preposition
Average sentence length	possessive pronoun
Number of characters	personal pronoun
Average word length	determiner
	adverb
	particle
	interjection

Table 3.1: Posit numeric output.

Posit was left untouched, however, a few adaptations had to be made. First

of all, when executing the Posit analysis a folder of results is created which is not overwritten when executing the next file leading to wrong outputs as previous intermediate results are included in new computations. As the tool was being applied sequentially on multiple texts, the calling script was embedded in a script which relocated the finished results and therefore cleaned intermediate ones. Another change involving the sentence count

Number of occurrences	Bi-gram
17	enterprise risk
9	synchrony financial
5	risk management
5	risk leader
5	email address
4	vp enterprise
3	your information
3	your email
3	use of

Table 3.2: Extract of Posit n-gram output.

was performed. After applying the tool on the first dataset, it became apparent that the sentence count was one smaller than it should have been. The script computing this count inserted new lines into the text in such a way that each sentence was stored in a single line with the aim to count the number of lines. However, after the last sentence no new line was inserted which lead to the final count being one too few. The bash command *awk '1'* was inserted which automatically inserts a new line at the end of the file if none is present.

Furthermore, many of the texts in the available datasets did not confirm to common linguistic rules such as having a space after a full stop followed by a capital letter. The script, however, only counted sentences if they were succeeded by at least one space. After changing the script to accept sentences followed by no space, the results of the Posit tool seemed to be more appropriate to the datasets. It needs to be mentioned that other faults are likely to occur due to this change, as e.g. web pages with capital letter are counted as several sentences (WWW.TestPage.COM). For the time being, errors like that have to be accepted when dealing with a large amount of web data, because content on web pages is more likely to disregard linguistic rules than other literature such as books or newspaper articles.

If the data is known to contain many instances that can not be split into sentences correctly with this approach, one should consider improving the Posit scripts to accomplish a better sentence count.

n-gram Reference Data

To broaden the set of features three 2-gram ratios are included in the research. The frequency in which certain 2-grams appear in a text can be calculated using the Posit tool. This, however, does not offer any information about how common a 2-gram is in the natural language. The aim is to create a value which gives an indication on how common 2-grams are in a single text compared to the whole English language. Before any ratios can be calculated a reference corpus is needed. The Google N-Gram corpus (Michel et al., 2011) contains around 217 000 000 bi-grams and was used for this dissertation. The 2-grams were stored in a database created with SQLite instead of text files to make accessing them as fast as possible, because requests have to be made for each text.
Based on Oakes (2003) it was chosen to calculate the *expected value* of each 2-gram appearing in both, the text and the reference corpus and then using their arithmetic mean. The expected value can be calculated using the following formula:

$$expected\ value = \frac{Row\ Total \cdot Column\ Total}{Overall\ Total} \tag{3.1}$$

Where *Row Total* is calculated as the sum of occurrences of a single 2-gram in the text and in the reference corpus, *Column Total* is defined as the sum of 2-grams in the text or in the reference and *Overall Total* is the sum of all row and column totals (see Table 3.3). The expected values of examples 'on the' is 37.96 and of 'is she' is 0.04, the mean is 19.

Bi-gram	Text	Reference	Row Total	Expected Value
on the	25	800328815	800328840	37.959717383
is she	13	849291	849304	0.040282617
Column Total	38	801178106	801178144	

Table 3.3: Calculation of expected value.

In this manner, three ratios were calculated: overall, low-frequency and high-frequency. For the low-frequency ratio only 2-grams with a lower

than average frequency in the reference corpus were considered for the calculations and equivalently for the high-frequency ratio only 2-grams with above average frequency. The overall ratio did not consider any differences in frequencies.

File Format

Included in the process of preparing the data is the decision in which format the datasets should be stored. Most of the analysis was done with the Weka tool and its Java API (Witten, Ian H and Frank and Hall, 2002). Weka's native file format is an Attribute Relation File Format (arff). It was decided to transform the data to this format as it is compatible with other programming languages such as R and Python. The data representation in arff files is separated into two parts: the header and the data.
Included in the header is global information about the type of data being stored. In arff files tags are used for notation such as *@RELATION*, *@ATTRIBUTE* and *@DATA*. The relation tag defines the name of the dataset and the attributes define the name and data type of features. Types can be numeric, nominal specification, string or dates. Nominal specifications are sets of strings which can be used to define e.g. the set of categories. Relation and attributes are defined in the header. After the tag @DATA each line describes one single instance where the attributes' values are comma separated in the order of definition in the header. An exemplary extract of an arff file looks as follows:

```
@RELATION    DBPEDIA

@ATTRIBUTE  ID   NUMERIC
@ATTRIBUTE  Average_sentence_length_(ASL)    NUMERIC
@ATTRIBUTE  Average_word_length_(AWL)    NUMERIC
@ATTRIBUTE  Number_of_characters     NUMERIC
@ATTRIBUTE  Number_of_sentences  NUMERIC
@ATTRIBUTE  Total_unique_words_(types)   NUMERIC
@ATTRIBUTE  Total_words_(tokens)     NUMERIC
@ATTRIBUTE  Type/Token_Ratio_(TTR)   NUMERIC
@ATTRIBUTE  adjective_types  NUMERIC
            ...
@ATTRIBUTE  2-gram_high_freq_ratio   NUMERIC
@ATTRIBUTE  2-gram_low_freq_ratio    NUMERIC
@ATTRIBUTE  category     {1,2,3,4}

@DATA
1,16.25,5.78462,376,4,51...1.0638297872340423,1.0833333333333333,1
2,22.6667,5.85294,398,3,54,...1.0508474576271185,1.057692307692308,1.0,1
3,17.5,6.42857,225,2,29,...0.9999999999999998,1.0000000000000002,1.0,1
4,18.5,5.91892,219,2,30,...1.0714285714285714,0.9999999999999999,1
```

As one can see in the example the attributes consist of the Posit features,

the n-gram ratios and the label. Additionally, an ID is introduced which refers to the line of the index file in which the path to the text file is stored or to the line in which the text was stored, depending on the dataset. This ID serves only as an identification of texts for the preprocessing and cleaning phase. Before feature selection and classification this attribute must be removed.

Application

In Appendix B.1 one can find the concrete changes applied to the Posit tool. The first code piece shows how the *oldSummaryFolder* and *oldNgramFolder* were moved to a new location that contained all summary and n-gram results for each text. The script can be executed on the current text file with the options –sum or –ngram depending on which result folder should be moved.
The second code fragment shows the script which counts the number of sentences. The only change applied was that in the *sed* command after a sentence delimiter (.?!) also zero spaces are accepted (\s*). Before, at least one space had to follow the end of the sentence (\s+ \s*).
The next two code pieces show how the tool was executed. First the Posit summary was created by calling *pos_all.sh* on the text file and then the script to move the results folder was executed. The same approach was taken to create the n-gram data with the *ngram.sh* script.

Appendix B.2 shows how the arff files were created by calling the Posit tool on every text.
The *run* method in file *main* is the start point of the process. The method is called with six arguments. The first argument determines which type of data is to be expected, the path defines where the data is stored, skip and count define the subset of files to transform, the id is used to separate the individual executions by naming the output files differently and relation is a variable defining the name of the arff relation. If the Drug data is to be transformed the *construct* method is called with the path to the data, the id of the execution, the name of the relation and any additional features and the class attributes. This constructor is then forwarded to the method that reads the files of the drug dataset. The same approach is taken for Extremism and DBpedia data, only the constructor is called with different class categories and no additional features.

The file also contains an *index* method that is used to create the index files for the Extremism and Drug data.

The *reader* file then handles the text files for the different datasets in separate methods. The DBpedia data was given in a csv file, therefore each line is parsed starting at the skip index and processing as many files as defined in the count variable. The category is read as the first element of the csv lines. The text is only further processed if it was correctly encoded, this means no '?' or other symbols dominate the text, and if it exceeds a certain size, to prevent Posit to be applied to empty files. A log file handles all files that are not further processes by saving the line number and the reason of ignoring the text. Afterwards the text is given to the constructor to apply the Posit tool.
The same principle is applied to the other datasets. The Extremism and Drug index files were read line by line. The Extremism files were stored in folders assigned to a category. Within these categories, the texts were stored in zip files which had to be extracted internally. Again the size of the file and the encoding were checked. The files also contained information irrelevant to this research which was ignored. As the Drug data was stored in separate folders as well index files were created for each folder. However, the folders were too large to be processed at once which is why a skip and count variable was introduced. As for the previous cases the size of the text was checked as well as the encoding and any other irrelevant lines were ignored. Moreover, only English language texts were accepted, which was detected using the langdetect library.
The *read_index* methods were used to create the index files for the datasets. For the Extremism data the indices were created for each category separately and for the Drug data the domain names were compared with a list of pre-classified drug related or unrelated domains. Then the categories *positive*, *negative* and *unknown* were directly assigned to the text files within the index files.

The *constructor* class manages the creation of several files: the text file, the resulting arff file and a log file which stores information about unprocessed texts. The class also contains a method called *execute* which calls the Posit shell scripts that execute the Posit summary calculations and the n-gram frequencies. Those results are processed by other classes and will be discussed later. Once all results are gathered in arrays the data can

be processed by the *util* class that writes the attributes and the data to temporary files. Whenever four new texts were processed the data was stored in a data file in order to follow the progress. The class contains other methods that handle writing, closing and removing temporary files.

The *create_arff* class handles the management of the attributes and their data values. The summary files are parsed in the *util* class and the results are stored in dictionaries where the keys are the attribute names and the values are the data. Three types of attributes are contained: the statistical data, the token and the POS data. The attributes are sorted alphabetically within their groups. The n-gram data is parsed in another method which forwards the results to the *ngram_ratio* class and returns the average, high and low frequency ratios.

The *util* file handles the parsing of the summary data by successively storing the statistical data, the tokens and the POS data in dictionaries. It also manages writing the tags for the arff file including the attribute names and the data types. Methods to write the data into files and to write the log files are included as well as the execution of bash scripts.

The *ngram* class in file *ngram_index* contains several dictionaries and the column and row sums necessary for calculating the n-gram frequency ratios as the row frequency multiplied with the column frequency and divided by the total sum. A reference threshold value is set, which is defined as the average frequency of 2-grams from the reference corpus. The column totals are calculated for the above and below threshold frequencies by updating the frequencies per column and by creating dictionaries which hold the frequency of the n-gram of the text and of the reference corpus. The class contains methods to search the database of the reference corpus and to parse the n-gram files created by Posit which contain a n-gram and its frequency in each line.

The *ngram_ratio* file contains methods to calculate the final ratio using the column and row frequency determined in the *ngram* class. These ratios are created for high, low and overall values of the frequencies. If n-grams are present in the text, but can not be found in the reference corpus, the frequencies are ignored by setting it to -1. These ratios are calculated for each individual n-gram which is why an average method is implemented that determines the average frequency and ignores the negative values as they mark non existing reference frequencies.

With the help of the *database* class the reference corpus was stored in a more efficient format. The Google n-gram data was given in separate files and is sorted alphabetically. Therefore a single database table was created holding the n-grams as text and their frequency as an integer. By iterating over each of those files every n-gram was added to the database. The database management was handled using the sqlite3 library. In order to find n-grams in the database the complete list of 2-grams in a text was searched in the database using the command 'SELECT * FROM database WHERE ngram IN (?,...,?)' with as many question marks as n-grams in the text. This command can handle up to 80 elements in the list. For texts with more than 80 n-grams the command was split up in multiple requests. A dictionary with keys for every n-gram is created that holds the frequencies found. If a n-gram could not be found in the database its value is -1. The class also contains a method *avg_freq* to calculate the average frequency over all 2-grams in the database in order to determine the threshold value between high and low frequency n-grams.

3.2.2 Data Processing

There are multiple machine learning tools including R and Python with its packages *pandas* and *sklearn* to only mention a few. However, this research was done using the Weka graphical user interface as well as its Java API, because the GUI offers an easy and quick way to get an overview of the distribution of instances across the 30 features offered by the visualisation interface. The API was utilised to implement the experiments and to print the results into files. Before the implementation of the experiments is presented it is important to get an image of the datasets. Afterwards, details such as cleansing, feature selection and classification models can be discussed.

The Datasets

In the following, three datasets are going to be presented. The source and topic of each dataset varies which provides a broad range of texts and will hopefully serve as proof that the Posit tool offers data which can be used efficiently for text classification.
Firstly, general information about the three datasets will be given and then

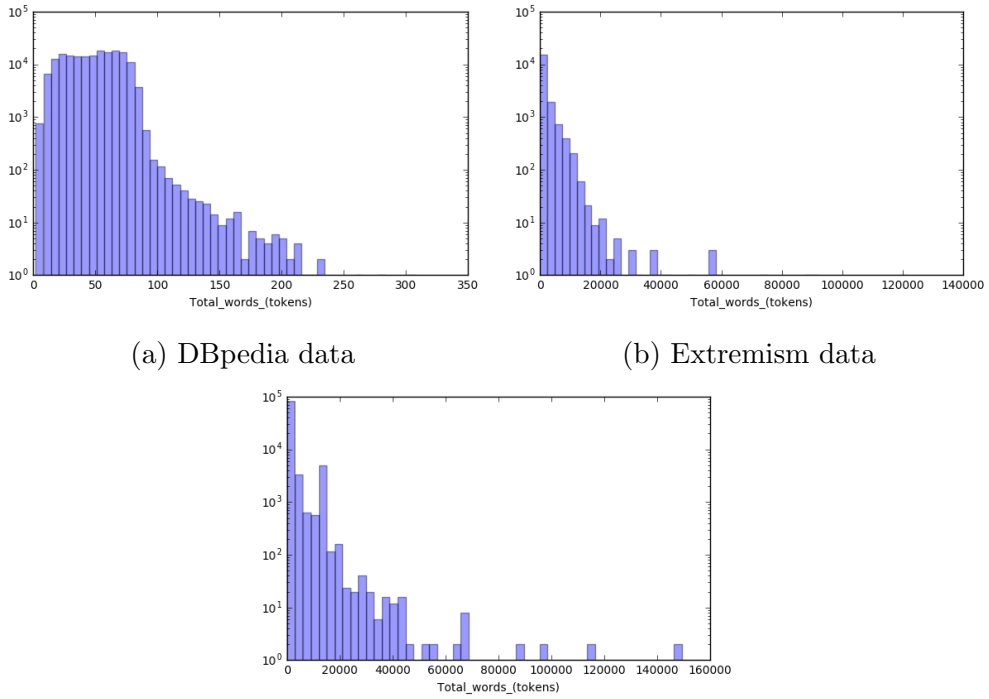
some of the statistical data from the Posit features will be compared to get a better image of what data is being analysed.

DBpedia The DBpedia data is a collection of Wikipedia articles extracted from DBpedia 2014. This corpus contains English texts covering 14 disjoint categories (Lehmann et al., 2015). Due to the large amount of data, it was decided to only analyse four classes: Company, Educational Institution, Artist and Athlete which were equally spread over 180 000 instances. After cleaning the data 179 972 instances were left.

Extremism This dataset has already been subjected to machine learning by Weir et al. (2016). The International CyberCrime Research Center (ICCRC) at Simon Fraser University in Burnaby, British Columbia developed a Terrorism and Extremism Network Extractor web-crawler (TENE) that searches the open web. Parts of the dataset extracted from TENE were used here. Before cleaning the dataset included 19 326 texts. After removing invalid files 18 464 remained. The whole data was pre-classified as positive, negative or neutral with 6925, 4836, 6703 instances respectively.

Drug A dataset containing drug related texts from the dark web was made available by ICCRC as well. The texts were gathered using the Dark Crawler (Mercur IT Solutions, 2017) originally developed by ICCRC. Parts of the data were manually classified by labelling the domains from which the texts were extracted as positive or negative: 120 domains were found to be drug related and 653 not drug related. Overall 1 245 410 texts were included, after cleaning 798 684 instances remained of which 91 088 are pre-classified.

As one can see in Table 3.4 the datasets show different distributions within the features *Total Words*, *Number of Sentences* and *Average Sentence Length*. The DBpedia set seems to contain mostly short texts as the mean of the number of total words is 48 and the maximum 308. On the other hand, the drug related data has an average of 1908 words in a text with a maximum of 149 371 and the extremism related set has an average of 1801 and a maximum of 121 324 words. The statistical data for all features of the three datasets can be found in Appendix A.
As one can see in Figure 3.1 the DBpedia data is nicely distributed with



(a) DBpedia data

(b) Extremism data

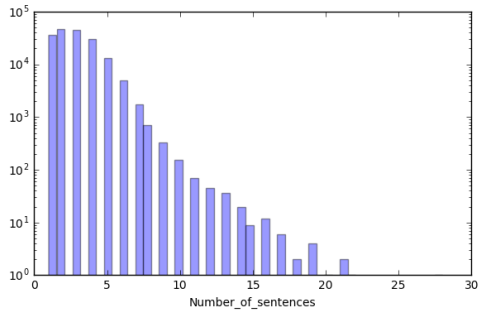
(c) Drug data

Figure 3.1: Distribution of number of words for all datasets.

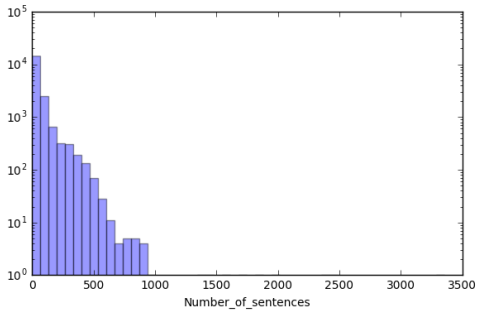
no extreme values, however, the extremism and drug data both show unusually large values which do not agree with the distribution of the rest of the data. Especially the drug data shows several unrelated occurrences over 60 000 words per text.

The number of sentences in a text also varies among those datasets. DBpedia contains on average 2.83 sentences, whereas the drug set contains about 100 times more: 271.38. The extremism data is with an average of 58 sentences closer to the DBpedia data than to the drug data.
In Figure 3.2 one can find histograms about the distribution of the number of sentences across the datasets. The maximum number of sentences for the extremism data is 3356, but the plot shows, that only a small amount of data includes that many sentences, as most of the data contains less than 1000 sentences. The drug data, on the other hand, is spread out with many samples around 5000 sentences per text and a few occurrences up to the maximum of 11 382 sentences.

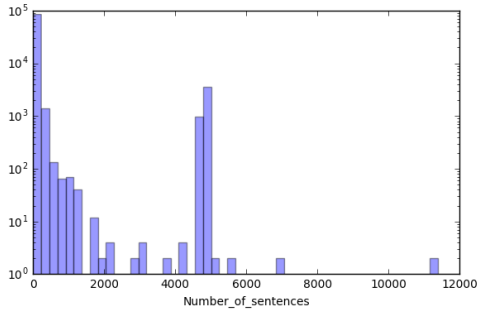
The values for average sentence length show that either the Posit calculations are not correct or the datasets contain many texts that are not structured in sentences. The mean of DBpedia's average sentence length is



(a) DBpedia data



(b) Extremism data

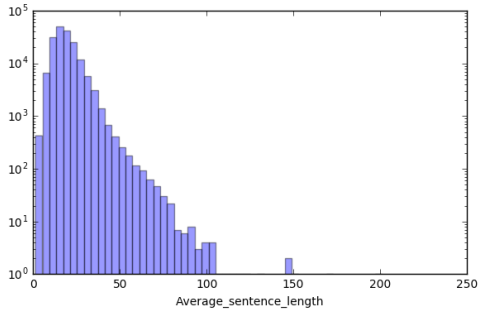


(c) Drug data

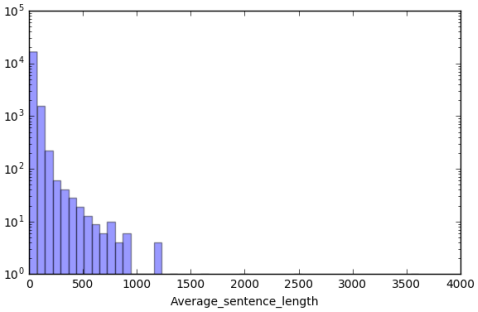
Figure 3.2: Distribution of number of sentences for all datasets.

18.58, which should be a reasonable length, but the maximum is 201. For the drug data the mean is 105.55 and the maximum is 66 894, which both are unlikely to be actual sentences. The extremism data's maximum is 5% of the maximum of the drug data, which is still unreasonably high. The average of 46 is still high, but closer to the DBpedia sentences.
In Figure 3.3 the distribution of the average sentence length is shown. As before, the DBpedia data shows the most compressed plot with only a few texts around 150. Most of the extremism values are located under 1000 with only one outlier at around 1200. Opposite to that is the drug data. Most of the instances are gathered between 1 and 15 000, but some data points can be found at up to 67 000.

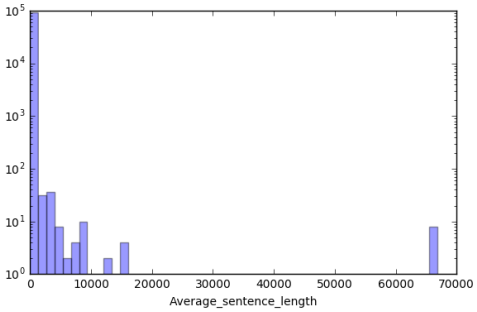
This comparison also shows that the range of features within one dataset varies too. For example in the DBpedia data the number of characters ranges between 10 and 1926 with a variance of 129, whereas the number of determiner types is in the interval from 0 to 7 with a variance of 0.75. This leads some evaluation metrics to assume that the number of characters feature has much more information, because the variance is much higher. To avoid this problem all datasets were scaled with two techniques.



(a) DBpedia data



(b) Extremism data



(c) Drug data

Figure 3.3: Distribution of average sentence length for all datasets.

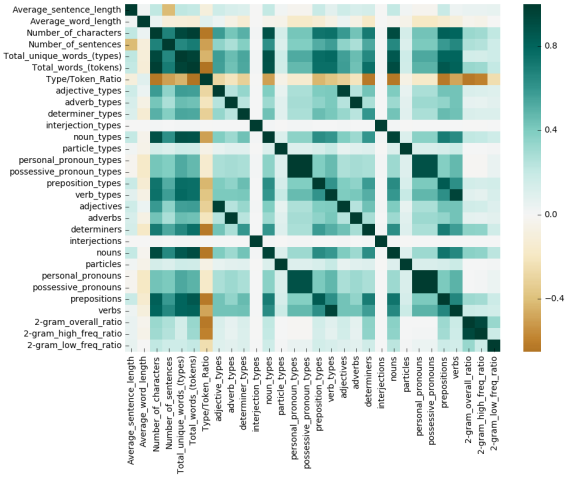
Feature	Dataset	Min	Max	Mean	Std Dev.
Total words	DBpedia	2	308	48.01	20.85
	Drug	3	149 371	1908.46	3729.39
	Extremism	2	121 324	1801.19	2880.69
Number of Sentences	DBpedia	1	28	2.83	1.49
	Drug	1	11 382	271.38	1064.49
	Extremism	1	3356	57.96	96.51
Average Sentence Length	DBpedia	1.33	201	18.58	7.18
	Drug	1.57	66 894	105.55	664.68
	Extremism	1.61	3612	46.35	66.63

Table 3.4: Statistics of datasets compared

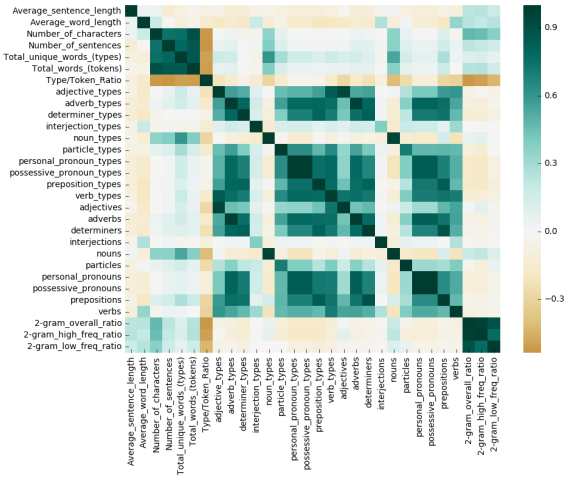
One version was normalised and the second was standardised (see Section 2.1.1) as it is not obvious which scaling technique works best on individual datasets.

Considering the correlation of features one can take a look at Figure 3.4 which shows a grid-like structure for each of the datasets. Uncorrelated features appear white in the figure and strongly correlating features are in

(a) DBpedia data



(b) Extremism data



(c) Drug data

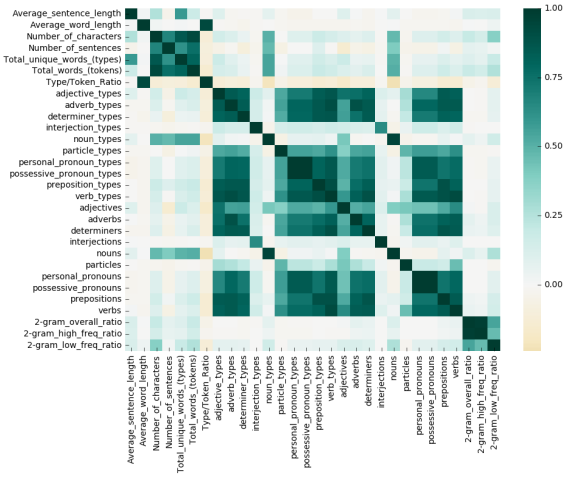


Figure 3.4: Correlation maps for all datasets.

green, if they are positively correlated and in brown if negatively correlated. One can see that the DBpedia data has more correlations within the feature set as the extremism or drug data, because the latter sets have lighter correlation maps. Furthermore, features such as number of characters and sentences, total unique words and total words are correlated to each other. The same behaviour can be found for personal and possessive pronouns and the 2-gram ratios, where especially the overall and high frequency ratios are correlated. Such correlations can have an impact on the machine learning algorithms, but the feature selection processes that were applied can reduce such behaviour.

Data Cleaning

The cleaning of data is an important process when using machine learning algorithms (Witten et al., 2011). Invalid or missing data can drastically reduce the models' prediction accuracy and needs to be avoided. Fortunately, the data created with the Posit tool does not return any missing values, however, as mentioned earlier, the Posit tool returned zero values for the feature *Number of Sentences* which lead to the conclusion that this value was not computed correctly. A similar case were faulty results for the 2-gram ratios which were -1. This indicated that no matches of 2-grams between text and reference data were found showing that the text was not correctly encoded or not in the English language. These erroneous instances were corrected by checking the language and the encoding in the preprocessing phase and computing the arff files again. This solved the problem of these zero or -1 values, but still left noise in the data. This noise, values that are inconsistent with the rest of the data, is harder to detect. One feature appearing to have faulty values is the *average sentence length* attribute. Even though the sentence count error in the Posit tool was fixed, the average sentence length is far too large for many cases across the three datasets as seen in Table 3.4. After analysing samples of data it emerged that many texts do not consist of normal sentence structures, but rather are a list of phrases. This might be due to the fact that the web crawler extracted all text elements from web pages, including buttons, fields, etc. and not only continuous text. The distribution of such values in the *average sentence length* might appear unusual and definitely does not agree with linguistics, however, such values can still be valuable for

categorisation as the extreme values might be more likely to appear in one category than in the other. For this reason those values were not discarded from the classification. Apart from these problems the implementation of the transformation process did not allow texts that did not fulfil criteria such as text size – a text must be at least 60 bytes, and proper encoding – the texts were decoded to UTF-8. Some of the texts were previously wrongly encoded and contained mostly punctuation characters such as '?' or spaces. Instances as such were ignored and not further processed.

Feature Selection

It was discussed previously that for high dimensional datasets it is important to reduce the features to an optimal subset to improve the accuracy and reduce the runtime of the classifier. This is not only necessary to eliminate correlated attributes but also because some classification models exploit the proximity between points for their calculations and in high dimensional space, the distance among instances becomes more uniform (Janecek et al., 2008). The feature selection process was applied after scaling the data, as differing ranges of features influences the calculations and might therefore distort the importance of features. The experiments were conducted on data that was normalised and standardised.
Of the three methods discussed in Section 2.1.2 Filter and PCA are going to be implemented and analysed. The wrapper approach was abandoned due to its computational runtime.

Filter methods The larger the set of features the more important it is to choose a good search algorithm in order to find the optimal feature subset. With k features the number of subsets grows exponentially to 2^k, therefore, an exhaustive search is not recommended for data with large k. Two heuristic searches are offered in the Weka API: Greedy Stepwise and Best First search, to be hereafter named GS and BS respectively.
GS is implemented as a steepest ascent Hill-Climber algorithm. By default the implementation of this algorithm starts with an empty set of features (forward search) and iteratively adds the best feature according to an evaluation metric. The algorithm terminates if it can not find a feature that improves the current set. It is called steepest ascent, as this version of Hill-Climbing takes into account the value of all successive nodes according to

the metric and does, therefore, not settle on the first feature that improves the current subset. The implementation also includes a backtracking ability which, by default, can backtrack up to five times before the search is terminated. The search offers three different search directions: forward, backward and bi-directional, which allows adding and removing features to optimise the sets effectiveness. The backward search starts with the full set of features and removes one by one depending on how much value each feature adds to the subset.

The implementation of BS is the Beam search, which is an optimised version of a best-first search. It is a greedy stepwise algorithm as it does not consider all possible features, but chooses the best option from a subset of possible features. Thus it is a memory efficient implementation of best-first, but does not always find the optimal solution.

GS and BS, being implemented as steepest ascent and beam search are both versions of the best-first search concept. The difference is that GS has three search directions: forward, backward and bi-directional, whereas BS can only search forwards or backwards.

In spite of being listed as a search algorithm, the Ranker does not return an optimal subset of features. It rather returns the list of features sorted by their value according to the evaluation metric. The user can decide either on a threshold down to which the ranked features should be listed, or on the number of features to be kept.

The evaluation metric used for the search and ranking methods is a vital part of feature selection. The Weka API offers the following attribute evaluators: OneR, CfsSubset, Correlation, GainRatio and InfoGain.

OneR chooses the attribute with the minimum error (Holte, 1993), by creating a rule that separates the data (Witten et al., 2011). CfsSubset evaluates the predictive power for the subset while keeping the correlations among this subset to a minimum (Hall, 1998). The Correlation attribute evaluation considers the Pearson's correlation between an attribute and the category. The Pearsons's correlation value is calculated as the covariance between attribute and the category divided by the product of their standard deviations (Guyon and Elisseeff, 2003):

$$Pearson(Class, Attribute) = \frac{cov(Class, Attribute)}{\sigma_{Class} \cdot \sigma_{Attribute}}$$

The category attribute is nominal in the datasets, in this case the evaluator

treats every nominal value as an indicator: when a class occurs it is treated as 1, if not as 0. The overall correlation is then a weighted average over the results per category.
To understand the information gain and gain ratio metrics one needs to define an entropy $H(C)$ which measures the information content of C (Witten et al., 2011). $H(Class)$ returns the information value of the category feature and $H(Class|Attribute)$ returns the expected value of information if the 'Attribute' is given.
Information gain calculates how much information is added when a feature is included:.

$$InfoGain(Class, Attribute) = H(Class) - H(Class|Attribute).$$

Similarly to that, the gain ratio is calculated between a feature and the category as the information gain divided by the information value of the attribute:

$$GainR(Class, Attribute) = \frac{H(Class) - H(Class|Attribute)}{H(Attribute)}$$

Six combinations between search methods and evaluation metrics are possible. GS and BS can only be combined with CfsSubset, because that is the only metric that takes sets of features into account. All other metrics can be combined with the Ranker method.
In order to compare the Ranker results with the feature subsets the highest ranked features were selected. The threshold down to which features are kept will be 80% of the rank of the best feature.

Principal Component Analysis To evaluate the last hypothesis from Section 3.1.1 the Pincipal Component Analysis has to be used on the set of features. It was mentioned earlier that the data has to be scaled before PCA can be applied, as PCA uses the variance of features to decide on their importance. If features have different scales, their variance can be disproportionally different. Two scaling options are available: normalising the data of each feature into the interval $[0, 1]$ or standardising the data such that the mean is zero and the standard deviation is one. Both options were applied to the datasets as there is no one scaling method most efficient for all datasets. Furthermore, when applying the PCA one needs to decide

how much of the overall variance in the dataset should be kept. If less variance of the data is kept fewer new features are necessary to account for that amount of variance. The default setting in Weka is 95% which was kept during the experiments.

Classification Models

Several machine learning algorithms come into question when classifying data. Four common implementations are going to be discussed in the following all of which are available in the Weka API: Naïve Bayes, Decision Tree, k-Nearest Neighbours and Support Vector Machines.

Naïve Bayes This algorithm is a probability based classification method. It works on the assumption that each features influences the probability of an instance belonging to a certain category independent of other features. It calculates the probability of a category k using conditional probabilities and the Bayes' theorem:

$$p(C_k|x) = \frac{p(C_k) \cdot p(x|C_k)}{p(x)} \tag{3.2}$$

where $x = (x_1, ...x_n)$ is the vector containing all n features (Witten et al., 2011, chap. 4).
Due to the assumption that features do not influence each other, this classifier should work best on datasets where the best features have been selected and therefore correlation is minimal.
The Weka implementation of the Naïve Bayes was set up with the kernel density estimator instead of the default normal distribution. This option was chosen, because the kernel estimator can deal better with non-Gaussian class distributions. Therefore this version is expected to perform better across all datasets.

Decision Tree This type of algorithm can be used for classification as well as regression problems. During the training phase a tree is constructed by starting at the root and choosing a feature which offers the best splitting point of the data. The branches of the root represent every possible value for this feature. Every node only contains instances following the branches from the root to this node. In each of the child nodes this process is

repeated until all instances in a node belong to the same category (Witten et al., 2011, chap. 4). The complete tree can then be pruned, which is a procedure that removes nodes or parts of the tree that do not offer much information towards the prediction. Pruning reduces the size of the tree and its complexity and thus the accuracy of the classifier improves as overfitting is less likely (Drazin, 2010).
The implementation chosed from Weka is the J48 decision tree with the standard settings, which are pruning with a confidence factor of 0.25 and at least two instances per leaf.

k-Nearest Neighbours In this method a new instance is classified by calculating its distance to the instances of the training set. The new data point is then labelled with the same class as the majority of its k-nearest neighbours. Feature selection should play an important role here, because proximities are more meaningful in less dimensional space (Cios et al., 2007; Janecek et al., 2008). This algorithm falls into the set of lazy learning algorithms as the calculations are postponed until classification. As this model considers only k many other instances it is sensitive to noisy data (Witten et al., 2011, chap. 4). For larger values of k this method is less sensitive to noise, but the distinctiveness among the classes is reduced.
The implementation in Weka is called IBk (instance based). It was executed with k=1 and k=3 using the LinearNNSearch of Weka combined with the euclidean distance.

Support Vector Machines Originally, this algorithm is used for binary classification problems only, but there exist variations applicable for multi-class problems. In this case the classification can be done for all pairs of categories or by splitting the classes in a way that one class competes with the rest. For the binary problem, this method tries to find a small set of instances on the border between classes, called support vectors. From these vectors, a function is build to maximise the distance between them. This function can have non-linear structures which results in multi-dimensional decision boundaries (Olson and Delen, 2008; Witten et al., 2011, chap. 4).
The Support Vector Machines (SVM) implementation in Weka follows the sequential minimal optimisation of Platt (1998) and is called SMO. It uses the pairwise binary representation for multi-class problems. The default settings were kept except for the scaling parameter. By default the classifier

works on the normalised data, but here it was chosen not to scale the data within the algorithm, but before as for all other classifiers.

Application

In Appendix B.3 the application for executing the text classification is shown. The implementation makes use of the Weka API (Witten, Ian H and Frank and Hall, 2002).
The *Execution* class represents the starting point of the application. The arguments include the data paths to the training and testing data, variables for the execution of centered or standardised PCA, which features should be deleted and an argument to decide if Posit or n-gram features are processed. The last variable is used to separate the produced files in different folders. From this *main* method the *Driver* is called over its two constructor methods.
The first constructor executes the PCA and feature selection. The training and testing data are processed by the *PrepareData* class to create objects compatible with the Weka API and to set the class attribute to the last feature. Then files for the PCA and the feature selection are created and both methods are executed.
The PCA transformed datasets are created by a method *pca* in the *SelectAttribute* class. The returned data is classified using the five machine learning methods.
The *executeFilter* method in the *Driver* class works on the already determined feature subsets. The inverse of the features of those sets are removed from the training and testing data before the five classifiers are applied to the data.

The other *Driver* constructor executes those five machine learning algorithms on training and testing data as well. It is possible to call this constructor with a list of features to delete before classifying the data. The five algorithms are called over the *RunClassifier* class which deals with the setting of options.

PrepareData is a class that manages the datasets. *createInstances* is a method that transforms the data source into an object of instances and sets the class attribute to the last feature in the list. *cleanData* is a method that, if necessary, transforms any non-nominal categories to a nominal data type, removes any features contained in a list or removes duplicates in the

dataset. The class contains methods to scale the data such as *standardise*, *normalise* or *center*. A *multiply* method is included multiplying all feature values with 10^{10} which is necessary for some filter methods. *remove* takes a list of feature indices and removes either those attributes or all attributes except for those in the list.

SelectAttributes is a class dealing with the feature selection methods discussed in this chapter: filter methods and PCA.
In method *filterAttributes* the filter methods are implemented which are combinations of evaluator and search methods stored in two lists. When iterating over those lists all possible combinations of filter methods are applied to the data set. The Weka class *AttributeSelection* can apply those filter methods to the data and returns the array of feature indices that are evaluated to be important. For the methods including the ranker search a threshold value is calculated as 80% of the best rank. Every feature that has a rank higher than this threshold is stored in a list. All gathered subsets are then returned to the *Driver* class.
To execute the PCA the Weka internal Principal Components filter is applied to the training data in method *pca*. Then the training and testing data is transformed using the new attributes and both datasets are returned to the *Driver* class.

The *RunClassifier* class includes a method *runClassifier* that applies a classifier to the training data and evaluates its accuracy on the testing data. Important results, such as the confusion matrix and the precision and recall are stored in the output files. For each machine learning model used in this dissertation a method was implemented that creates an object of the model and sets the preferred options.

Chapter 4

Analysis

After implementing all experiments and executing the analysis on all datasets available with different feature sets there are now many results that need to be discussed. To simplify the presentation of the results the attributes will be described by their ID hereafter. The list of all attributes and IDs can be found in Table 4.1. Identifier 0 - 26 originate from the Posit summary data and attributes 27 - 29 are the 2-gram ratios calculated from the 2-gram frequency.

ID	Attributes	ID	Attributes
0	average sentence length	15	preposition types
1	average word length	16	verb types
2	number of characters	17	adjectives
3	number of sentences	18	adverbs
4	total unique words (types)	19	determiners
5	total words (tokens)	20	interjections
6	type/token ratio	21	nouns
7	adjective types	22	particles
8	adverb types	23	personal pronouns
9	determiner types	24	possessive pronouns
10	interjection types	25	prepositions
11	noun types	26	verbs
12	particle types	27	2-gram overall ratio
13	personal pronoun types	28	2-gram high freq ratio
14	possessive pronoun types	29	2-gram low freq ratio

Table 4.1: Attributes with their ID

Furthermore, as there are six versions of filter methods they were named S1 to S6 as can be seen in Table 4.2. The first four are combinations with

the ranker method and S5 and S6 are greedy stepwise and best first search respectively.

Search	Evaluator	Filter Method
Ranker	OneR	S1
	Correlation	S2
	Gain Ratio	S3
	Information Gain	S4
Greedy Stepwise	CFS subset	S5
Best First	CFS subset	S6

Table 4.2: Filter Methods.

In the first section of this chapter the results of the experiments will be presented for each of the five hypotheses. For each experimental setting the precision and recall of the classifier was recorded. The values for precision and recall were calculated as explained in Section 2.1.4 and suggested by Sebastiani (2002).
Subsequently, having gathered all results, it will be discussed how they agree with the expectations formalised in hypotheses in Section 3.1.1. Furthermore, statements about the efficiency of scaling techniques and machine learning algorithms can be made considering all three datasets.

4.1 Results

The five classifiers that were used to evaluate the experiments are J48, Naïve Bayes, k-Nearest Neighbours with k=1 and k=3 (kNN1, kNN3) and SVM. The three datasets were either normalised or standardised before classification.
In the following the results of the experiments will be presented for each dataset. First, the five classifiers were used on both Posit features and all features to see how the 2-gram ratios influence the classification results as discussed in Hypothesis 1. Secondly, the Posit features were combined with the low-frequency and then with the high-frequency 2-gram ratio (see Hypothesis 2). Thirdly, the process was repeated on only the 2-gram ratios (Hypothesis 3) to evaluate how well those data values can be classified. Fourthly, the previously mentioned filter methods (see Table 4.2) and their resulting subsets were evaluated. Lastly, the learning methods were applied to new feature sets created by the PCA.

4.1.1 Hypothesis 1

To asses the value of the 2-gram ratios combined with the Posit features both feature sets were used for classification on the five machine learning models. The data was used in two settings: normalised and standardised to find out which offers better results. In all tables presenting the results of this experiment the best F_1 values for each machine learning model are marked grey and the best overall result is marked green.

DBpedia

The results for the DBpedia data can be found in Table 4.3. It can be easily spotted that the best classifications were achieved using standardised data. For that dataset SVM delivered the best classification for the full feature set on standardised data, which has a F_1 value of 0.557. Overall the standardised data lead to better predictions than the normalised data. J48 and SVM turned out to be models with a worse predictive power on the normalised data as their precision and recall values vary. When considering the confusion matrix it becomes apparent that J48 did not predict the Athlete class at all and SVM did not predict Artist nor Athlete. For all other classifiers precision and recall are more balanced.

Alg.	Scale	Posit			Posit + 2-gram		
		Precision	Recall	F_1	Precision	Recall	F_1
J48	Norm	0.196	0.336	0.248	0.161	0.221	0.186
	Std.	0.496	0.497	0.496	0.497	0.497	0.497
Naïve Bayes	Norm	0.244	0.264	0.254	0.215	0.226	0.220
	Std.	0.464	0.447	0.455	0.472	0.463	0.467
kNN1	Norm	0.244	0.239	0.241	0.322	0.280	0.300
	Std.	0.465	0.466	0.465	0.469	0.469	0.469
kNN3	Norm	0.23	0.257	0.243	0.306	0.310	0.308
	Std.	0.497	0.483	0.490	0.500	0.486	0.493
SVM	Norm	0.104	0.267	0.150	0.074	0.252	0.114
	Std.	0.551	0.552	0.551	0.557	0.557	0.557

Table 4.3: Classification results for normalised and standardised DBpedia data for Posit and Posit + 2-gram attributes.

Extremism

The classifications of the extremism data showed the best results when using the full set of Posit and 2-gram features with the exception of J48 and kNN3 which showed a 0.002 and 0.001 larger F_1 value respectively when using only Posit features. SVM is the only algorithm that performed better on standardised than on normalised data. In total, the best classification was achieved by kNN1 on the normalised data with the full feature set of Posit and 2-gram values. All results can be found in Table 4.4

Alg.	Scale	Posit			Posit + 2-gram		
		Precision	Recall	F_1	Precision	Recall	F_1
J48	Norm	0.959	0.959	0.959	0.957	0.957	0.957
	Std.	0.914	0.914	0.914	0.918	0.918	0.918
Naïve Bayes	Norm	0.703	0.703	0.703	0.712	0.711	0.711
	Std.	0.69	0.688	0.689	0.693	0.690	0.691
kNN1	Norm	0.964	0.964	0.964	0.965	0.965	0.965
	Std.	0.912	0.913	0.912	0.912	0.912	0.912
kNN3	Norm	0.921	0.921	0.921	0.920	0.920	0.920
	Std.	0.901	0.901	0.901	0.901	0.901	0.901
SVM	Norm	0.761	0.761	0.761	0.768	0.767	0.767
	Std.	0.78	0.777	0.778	0.789	0.787	0.788

Table 4.4: Classification results for normalised and standardised Extremism data for Posit and Posit + 2-gram attributes.

Drug

The Drug dataset accomplished the best results for the normalised data. However, the distinction between the feature sets is not that clear. As one can see in Table 4.5 for J48, kNN1 and kNN3 the results of both feature sets on the normalised data are the same. Only Naïve Bayes performs better on the Posit features with a F_1 value increased by 0.36, than on the full set of features. The best F-measure of 0.995 resulted from kNN1, closely followed by J48 with an F-measure of 0.99.
The standardised dataset also shows differences between precision and recall for J48 on the Posit feature set, where mostly the negative class was predicted.

Alg.	Scale	Posit			Posit + 2-gram		
		Precision	Recall	F_1	Precision	Recall	F_1
J48	Norm	0.99	0.99	0.99	0.99	0.99	0.99
	Std.	0.645	0.425	0.512	0.346	0.392	0.368
Naïve Bayes	Norm	0.818	0.814	0.816	0.794	0.767	0.780
	Std.	0.690	0.602	0.643	0.683	0.599	0.638
kNN1	Norm	0.995	0.995	0.995	0.995	0.995	0.995
	Std.	0.465	0.421	0.442	0.463	0.419	0.440
kNN3	Norm	0.979	0.979	0.979	0.979	0.979	0.979
	Std.	0.430	0.397	0.413	0.432	0.400	0.415
SVM	Norm	0.878	0.868	0.873	0.879	0.869	0.874
	Std.	0.678	0.608	0.641	0.682	0.611	0.645

Table 4.5: Classification results for normalised and standardised Drug data for Posit and Posit + 2-gram attributes.

4.1.2 Hypothesis 2

The aim of this hypothesis is to detect if there is a difference in efficiency between the low and high frequency 2-gram ratio. It was suggested that the low frequency ratio tend to be more important, because less frequent word combinations might be more distinctive among different categories. The best results per machine learning algorithm were marked grey and the result of the most accurate classifier was marked green.

DBpedia

The results for this dataset show that the difference in the feature sets did not have much impact on the classification as the results for Posit + low frequency ratio are almost the same as for Posit + high frequency (see Table 4.6). For normalised data the average for the low frequency ratio is 0.217 and for the feature set containing the high frequency ratio it is 0.216. For the standardised data low and high frequency ratios achieved 0.552 and 0.555 respectively. Therefore, the standardised data served as a better basis for classification. Except for the standardised versions of Naïve Bayes, kNN1, kNN3 and SVM all results are equal for the two feature subsets. For those cases the high frequency ratio achieved slightly better results. The overall best result was accomplised by SVM on standardised data which had a result of 0.555 for the F_1 value.

When considering the classification results of the normalised data it stands out that precision and recall have larger gaps between each other for the same models. Taking a closer look at the confusion matrix shows that J48 and SVM mostly predicted categories Company and Educational Institution, but not Artist or Athlete, which influenced the recall to be higher than the precision. For higher precision and recall values the differences are negligible, showing a more stable model.

Alg.	Scale	Posit + low freq.			Posit + high freq.		
		Precision	Recall	F_1	Precision	Recall	F_1
J48	Norm	0.128	0.253	0.170	0.128	0.253	0.170
	Std.	0.495	0.496	0.495	0.495	0.496	0.495
Naïve Bayes	Norm	0.207	0.226	0.216	0.207	0.226	0.216
	Std.	0.469	0.446	0.457	0.471	0.458	0.464
kNN1	Norm	0.271	0.277	0.274	0.271	0.277	0.274
	Std.	0.465	0.466	0.465	0.467	0.468	0.467
kNN3	Norm	0.263	0.289	0.275	0.263	0.289	0.275
	Std.	0.496	0.482	0.489	0.499	0.485	0.492
SVM	Norm	0.101	0.264	0.146	0.101	0.264	0.146
	Std.	0.552	0.553	0.552	0.556	0.555	0.555

Table 4.6: Classification results for normalised and standardised DBpedia data for Posit + low frequency 2-gram ratio and Posit + high frequency 2-gram attributes.

Extremism

Throughout this dataset the results of precision and recall are the same for both feature subsets. The normalised data returned better results than the standardised data, except for SVM where the F_1 value for standardised data increased by 0.017. Furthermore, kNN1 achieved with 0.965 the best F_1 value closely followed by J48 with 0.955. The full comparison can be found in Table 4.7

Drug

The Drug data behaves similarly to the Extremism data. The values of precision and recall do not depend on the feature subset. The kNN1 algorithm achieved the highest F_1 value with 0.995 followed by J48 with 0.990.

Alg.	Scale	Posit + low freq.			Posit + high freq.		
		Precision	Recall	F_1	Precision	Recall	F_1
J48	Norm	0.955	0.955	0.955	0.955	0.955	0.955
	Std.	0.917	0.917	0.917	0.917	0.917	0.917
Naïve Bayes	Norm	0.705	0.705	0.705	0.705	0.705	0.705
	Std.	0.692	0.690	0.691	0.692	0.69	0.691
kNN1	Norm	0.965	0.965	0.965	0.965	0.965	0.965
	Std.	0.913	0.913	0.913	0.913	0.913	0.913
kNN3	Norm	0.920	0.920	0.920	0.92	0.92	0.920
	Std.	0.902	0.902	0.902	0.902	0.902	0.902
SVM	Norm	0.766	0.765	0.765	0.766	0.765	0.765
	Std.	0.784	0.781	0.782	0.784	0.781	0.782

Table 4.7: Classification results for normalised and standardised Extremism data for Posit + low frequency 2-gram ratio and Posit + high frequency 2-gram attributes.

Overall the normalised data performed better than the standardised data. The best result for the standardised data is only 0.662 for SVM on both feature subsets.
As for the DBpedia data, J48 and SVM show fluctuations between the precision and recall values of the standardised data. For both J48 models the negative category was favoured, whereas for SVM the positive class was predicted more often.

4.1.3 Hypothesis 3

The objective of this hypothesis was to find out how much valuable information is contained in the 2-gram features. It was expected that the machine learning algorithms do not perform as well as if the Posit features were included. For each algorithm the best F_1 value was marked grey to highlight any differences between standardised and normalised data. The best result overall was marked green.

DBpedia

The F_1 values for the five machine learning models are overall quite poor, the highest being 0.364 for SVM applied to standardised data (see Table 4.9). The average performance over all classifiers applied to normalised data

Alg.	Scale	Posit + low freq. Precision	Recall	F_1	Posit + high freq. Precision	Recall	F_1
J48	Norm	0.990	0.990	0.990	0.99	0.99	0.990
	Std.	0.645	0.425	0.512	0.645	0.425	0.512
Naïve Bayes	Norm	0.822	0.818	0.820	0.822	0.818	0.820
	Std.	0.632	0.626	0.629	0.632	0.626	0.629
kNN1	Norm	0.995	0.995	0.995	0.995	0.995	0.995
	Std.	0.463	0.419	0.440	0.463	0.419	0.440
kNN3	Norm	0.979	0.979	0.979	0.979	0.979	0.979
	Std.	0.432	0.400	0.415	0.432	0.4	0.415
SVM	Norm	0.879	0.869	0.874	0.879	0.869	0.874
	Std.	0.758	0.588	0.662	0.758	0.588	0.662

Table 4.8: Classification results for normalised and standardised Drug data for Posit + low frequency 2-gram ratio and Posit + high frequency 2-gram attributes.

is 0.184 and applied to standardised data is 0.351 which shows how little information was contained in this feature set. Over the whole experiment standardised data achieved higher results than the normalised data.

Algorithm	Scale	2-gram Features Precision	Recall	F_1
J48	Norm	0.262	0.251	0.256
	Std.	0.358	0.369	0.363
Naïve Bayes	Norm	0.173	0.236	0.200
	Std.	0.320	0.351	0.335
kNN1	Norm	0.139	0.249	0.178
	Std.	0.341	0.348	0.344
kNN3	Norm	0.141	0.268	0.185
	Std.	0.343	0.351	0.347
SVM	Norm	0.063	0.251	0.101
	Std.	0.372	0.357	0.364

Table 4.9: Classification results for normalised and standardised DBpedia data only with the 2-gram features.

Extremism

The results of the extremism data using only 2-gram features can be found in Table 4.10. One can see that the F_1 value is the highest for kNN1 (0.885) followed by J48 (0.812) both applied to normalised data . Except for SVM, the normalised data accomplished better results than the standardised data. On normalised data an average F_1 value of 0.688 was achieved and 0.627 on standardised data showing the scaling method did not effect the outcome of the experiment much.

Algorithm	Scale	2-gram Features		
		Precision	Recall	F_1
J48	Norm	0.812	0.813	0.812
	Std.	0.699	0.694	0.696
Naïve Bayes	Norm	0.539	0.54	0.539
	Std.	0.518	0.512	0.515
kNN1	Norm	0.885	0.885	0.885
	Std.	0.717	0.717	0.717
kNN3	Norm	0.775	0.775	0.775
	Std.	0.697	0.695	0.696
SVM	Norm	0.431	0.423	0.427
	Std.	0.491	0.529	0.509

Table 4.10: Classification results for normalised and standardised Extremism data only with the 2-gram features.

Drug

The best classification results using only the 2-gram ratios were accomplished on the Drug data, as one can see in Table 4.11. On average the F_1 value is 0.731 for normalised data and 0.589 for standardised data. For Naïve Bayes the standardised data worked better, for SVM both worked equally poorly. The rest of the algorithms performed better on normalised data. The best result of 0.937 was recorded for kNN1 on normalised data, followed by J48 with 0.875 and kNN3 with 0.874.

Whereas most precision and recall values are balanced, SVM on the standardised data shows a large gap between the higher recall (0.588) and the lower precision (0.346). This is due to the fact that the SVM model did not

predict the negative class at all, resulting in a perfect recall for the positive class, but also leading to zero precision and recall for the negative class.

Algorithm	Scale	2-gram Features		
		Precision	Recall	F_1
J48	Norm	0.877	0.874	0.875
	Std.	0.631	0.575	0.602
Naïve Bayes	Norm	0.494	0.581	0.534
	Std.	0.661	0.65	0.655
kNN1	Norm	0.939	0.936	0.937
	Std.	0.64	0.594	0.616
kNN3	Norm	0.875	0.873	0.874
	Std.	0.649	0.622	0.635
SVM	Norm	0.346	0.588	0.436
	Std.	0.346	0.588	0.436

Table 4.11: Classification results for normalised and standardised Drug data only with the 2-gram features.

4.1.4 Hypothesis 4

Following on from Section 2.1.2 where feature selection was discussed, this hypothesis revolves around analysing which feature subsets were the most efficient. The five algorithms were applied to normalised and standardised data as before.

For each dataset first the subsets found using the filter methods from Table 4.2 will be presented, followed by the classification results of each of those feature subsets.

For subsets S1 - S4 the ranker method was used which does not select a subset but returns a sorted list of all features with decreasing importance for the classification. In these experiments the subsets were created by keeping only those features with a rank of more than 80% of the best feature. This results in small subsets if the rank decreases rapidly or in large subsets if the importance does not vary a lot.

Grey cells mark the best results within a feature subset and green cells mark the best classification overall.

DBpedia

Looking at Table 4.12 it is easily noticeable that the subsets are equal for normalised and standardised data. Furthermore, S2 and S4 (Correlation and Information Gain) kept only one feature which is the Type/Token Ratio. Every other subset includes at least one 2-gram ratio. The subset sizes are 1, 4, 7 and 15.

Filter Method	Scale	Attribute Subset
S1	Norm Std.	1, 4, 5, 6, 13, 14, 16, 23, 24, 25, 26, 27, 28
S2	Norm Std.	6
S3	Norm Std.	6, 13, 14, 23, 24, 27, 28
S4	Norm Std.	6
S5	Norm Std.	1, 6, 13, 23, 27, 28, 29
S6	Norm Std.	1, 6, 23, 27

Table 4.12: Subset results for normalised and standardised DBpedia data

As mentioned before, S2 and S4 developed exactly the same feature subset, which is why S4 is not included in Table 4.13. For this data the best classifications were found by standardising the data. Except for feature set S2 (and S4) SVM accomplished the best results. For S2 kNN1 and kNN3 performed better. The best result was computed using subset S1 on standardised data using SVM. However, this results is only 0.515 and 0.514 for precision and recall respectively.
Several unbalanced predictions can be spotted in Table 4.13 all occurring for the normalised data. For subsets S1 to S4, J48 has a much larger recall than precision, as does SVM on subsets S2 to S4. On subset S5 and S6 kNN1 and kNN3 have better precision values than recall. These irregularities are induced by unbalanced predictions among the classes.

Subset	Algorithm	Normalised		Standardised	
		Precision	Recall	Precision	Recall
S1	J48	0.197	0.336	0.485	0.487
	Naïve Bayes	0.200	0.216	0.478	0.469
	kNN1	0.272	0.223	0.430	0.430
	kNN3	0.285	0.271	0.465	0.454
	SVM	0.206	0.241	0.515	0.514
S2 & S4	J48	0.063	0.252	0.384	0.390
	Naïve Bayes	0.250	0.220	0.381	0.382
	kNN1	0.177	0.209	0.392	0.396
	kNN3	0.291	0.262	0.392	0.396
	SVM	0.062	0.248	0.350	0.371
S3	J48	0.202	0.334	0.453	0.454
	Naïve Bayes	0.203	0.219	0.407	0.403
	kNN1	0.279	0.236	0.408	0.402
	kNN3	0.277	0.304	0.427	0.416
	SVM	0.174	0.279	0.458	0.452
S5	J48	0.363	0.305	0.478	0.479
	Naïve Bayes	0.194	0.232	0.436	0.409
	kNN1	0.361	0.253	0.411	0.410
	kNN3	0.511	0.251	0.436	0.428
	SVM	0.279	0.251	0.493	0.488
S6	J48	0.363	0.305	0.486	0.485
	Naïve Bayes	0.194	0.19	0.432	0.412
	kNN1	0.334	0.236	0.409	0.408
	kNN3	0.262	0.238	0.435	0.429
	SVM	0.313	0.258	0.489	0.484

Table 4.13: Classifier results for normalised and standardised DBpedia data; for normalised and standardised data S2 and S4 are equal.

Extremism

For this dataset the subsets mostly vary between normalised and standardised data. Most of the subsets only differ in one feature: for S1 the subset for the standardised data included attribute 9 (determiner types); for S2 the set for normalised data includes feature 8 (adverb types); for Set 3 the standardised setting includes attribute 28 (2-gram high freq ratio); for S4 the subsets are equal; for S5 the normalised data set includes features 6 and 12 (Type/Token Ratio and particle types); the feature sets for S6 are equal. It should be pointed out that the elements 11 and 21 (nouns and

noun types) are included in all twelve feature subsets.
S1 is by far the largest set including 25 and 26 features. The rest of the subsets range between 3 and 9 elements. The list of all features for the different settings can be seen in Table 4.14.

Filter Method	Scale	Attribute Subset
S1	Norm	0, 1, 2, 3, 4, 5, 6, 7, 8, 11, 13, 14, 15, 16, 17, 18, 19, 21, 23 , 24, 25, 26, 27, 28, 29
	Std.	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 13, 14,15, 16, 17, 18, 19, 21, 23, 24, 25, 26, 27, 28, 29
S2	Norm	8, 11, 13, 14, 18, 21, 23, 24
	Std.	11, 13, 14, 18, 21, 23, 24
S3	Norm	1, 11, 21
	Std.	1, 11, 21, 28
S4	Norm Std.	1, 11, 21
S5	Norm	1, 6, 11, 12, 13, 18, 21, 27, 28
	Std.	1, 11, 13, 18, 21, 27, 28
S6	Norm Std.	1, 11, 13, 18, 21, 27, 28

Table 4.14: Subset results for normalised and standardised Extremism data.

It is important to remember that even if the subsets of normalised and standardised data are equal, the classification results can still differ. For the normalised data S3 and S4 are equal and for the standardised data S5 and S6, which is why some results were left out of Table 4.15. Other than for the DBpedia data, the extremism corpus was classified best when using the normalised data. Throughout the experiment kNN1 performed best closely followed by J48. Overall feature set S1 outperformed the rest with 0.968 precision and recall only marginally better than the results of J48 on the same feature set (0.957 and 0.956).

Drug

As for the other datasets there is much resemblance between feature subsets found for normalised and for standardised data. For this data however, the subsets are exactly the same for both settings. Apart from S1 none of

Subset	Algorithm	Normalised		Standardised	
		Precision	Recall	Precision	Recall
S1	J48	0.957	0.956	0.917	0.917
	Naïve Bayes	0.706	0.705	0.691	0.689
	kNN1	0.968	0.968	0.913	0.913
	kNN3	0.928	0.928	0.899	0.899
	SVM	0.769	0.768	0.784	0.783
S2	J48	0.901	0.901	0.821	0.821
	Naïve Bayes	0.644	0.646	0.645	0.646
	kNN1	0.935	0.935	0.833	0.833
	kNN3	0.861	0.862	0.818	0.818
	SVM	0.597	0.581	0.622	0.621
S3	J48	0.904	0.905	0.857	0.858
	Naïve Bayes	0.703	0.686	0.713	0.685
	kNN1	0.929	0.929	0.831	0.832
	kNN3	0.848	0.848	0.812	0.812
	SVM	0.419	0.535	0.537	0.545
S4	J48			0.839	0.840
	Naïve Bayes			0.704	0.680
	kNN1			0.814	0.815
	kNN3			0.796	0.795
	SVM			0.545	0.552
S5	J48	0.939	0.938	0.880	0.880
	Naïve Bayes	0.724	0.721	0.708	0.700
	kNN1	0.947	0.947	0.844	0.845
	kNN3	0.882	0.881	0.829	0.829
	SVM	0.653	0.649	0.595	0.591
S6	J48	0.932	0.932		
	Naïve Bayes	0.706	0.702		
	kNN1	0.940	0.940		
	kNN3	0.872	0.872		
	SVM	0.616	0.596		

Table 4.15: Classifier results for normalised and standardised Extremism data; S4 is equivalent to S3 for normalised data and S6 is equivalent to S5 for standardised data.

the other sets include 2-gram features. S1 however, includes all of them and is with 17 elements, the largest subset found. The other subsets range between two and six elements and all of them include features 11 and 21 (nouns and noun types) as it was already pointed out for the Extremism

data. Filter methods S5 and S6 found the same subsets of important features.

Filter Method	Scale	Attribute Subset
S1	Norm Std.	0, 1, 2, 3, 4, 5, 6, 11, 16, 17, 19, 21, 25, 26, 27, 28, 29
S2	Norm Std.	11, 21
S3	Norm Std.	2, 11, 21
S4	Norm Std.	2, 5, 11, 21
S5	Norm Std.	0, 2, 4, 11, 21, 22
S6	Norm Std.	0, 2, 4, 11, 21, 22

Table 4.16: Subset results for normalised and standardised Drug data.

The best results for the filtered feature subsets were found for the normalised data. For all subsets the best classifications were achieved using a k-Nearest Neighbour classifier with k=1. J48 and kNN3 also reached high precision and recall values of over 0.95. The best classification was found for the S1 feature set which reached a precision and recall of 0.994.
Even though normalised and standardised data worked on the same feature sets, the classifiers applied to standardised data were outperformed by every model on the normalised data.
For the standardised data the classification models were evaluated to be unstable across several feature subsets and several machine learning algorithms. These unbalances are due to the fact that for most cases only one of the two categories was predicted.

4.1.5 Hypothesis 5

A principal component analysis was applied to the normalised and standardised data and was executed in two settings: centered and standardised. This means the normalised data was both centered and standardised and standardised data was centered and kept standardised.

Subset	Algorithm	Normalised		Standardised	
		Precision	Recall	Precision	Recall
S1	J48	0.990	0.990	0.169	0.411
	Naïve Bayes	0.783	0.727	0.751	0.637
	kNN1	0.994	0.994	0.725	0.591
	kNN3	0.978	0.978	0.737	0.591
	SVM	0.867	0.853	0.723	0.591
S2	J48	0.950	0.949	0.169	0.412
	Naïve Bayes	0.852	0.837	0.346	0.588
	kNN1	0.969	0.969	0.169	0.412
	kNN3	0.944	0.944	0.771	0.489
	SVM	0.784	0.779	0.346	0.588
S3	J48	0.969	0.969	0.169	0.412
	Naïve Bayes	0.842	0.827	0.346	0.588
	kNN1	0.986	0.986	0.169	0.412
	kNN3	0.953	0.953	0.169	0.412
	SVM	0.799	0.795	0.346	0.588
S4	J48	0.978	0.978	0.169	0.412
	Naïve Bayes	0.840	0.824	0.346	0.588
	kNN1	0.988	0.988	0.169	0.412
	kNN3	0.955	0.956	0.169	0.412
	SVM	0.801	0.796	0.346	0.588
S5 & S6	J48	0.983	0.983	0.329	0.391
	Naïve Bayes	0.823	0.816	0.758	0.589
	kNN1	0.989	0.989	0.169	0.412
	kNN3	0.960	0.960	0.169	0.412
	SVM	0.835	0.826	0.737	0.600

Table 4.17: Classifier results for normalised and standardised Drug data; S5 is equivalent to S6 for normalised and standardised data.

The number of features was reduced in every setting. For the DBpedia data the features were reduced to ten for the normalised data on the centered PCA and 13 for every other setting. The features of the Extremism set were decreased to nine for the normalised and centered approach and 14 for the other settings. The Drug data shows the most extreme reduction to only one feature for the standardised and centered PCA, seven for the normalised and centered PCA and 13 for both standardised methods.
For this hypothesis it was chosen to also calculate the F-measure as precision and recall differ in some cases, such that the F_1 value gives a clearer

view on how well the classifiers performed. Grey cells mark the best results within a machine learning algorithm and green cells mark the best classification overall.

DBpedia

The best values for the F-measure for each algorithm were highlighted in Table 4.18. Throughout this experiment the DBpedia data was better classified when standardised instead of normalised. The best classification was performed by SVM, where centered and standardised PCA performed with an F_1 value of 0.5 equally well. The worst classification also resulted from SVM applied to the normalised data with a value of 0.099 for centered and 0.174 for standardised data. Overall the centered PCA achieved better results as the standardised PCA, only for J48 and SVM the F-measure was equal.
Throughout this experiment on normalised data the classifiers have little predictive power as not all categories were predicted. Especially J48, kNN1 and SVM are unbalanced among the predicted classes leading to differences in precision and recall.

Alg.	Scale	Centered			Standardised		
		Precision	Recall	F_1	Precision	Recall	F_1
J48	Norm	0.187	0.239	0.210	0.185	0.255	0.214
	Std.	0.462	0.462	0.462	0.461	0.463	0.462
Naïve Bayes	Norm	0.239	0.240	0.239	0.249	0.247	0.248
	Std.	0.478	0.474	0.476	0.477	0.473	0.475
kNN1	Norm	0.178	0.235	0.203	0.146	0.251	0.185
	Std.	0.449	0.449	0.449	0.445	0.445	0.445
kNN3	Norm	0.219	0.208	0.213	0.275	0.261	0.268
	Std.	0.481	0.468	0.474	0.479	0.465	0.472
SVM	Norm	0.062	0.249	0.099	0.130	0.263	0.174
	Std.	0.502	0.499	0.500	0.502	0.499	0.500

Table 4.18: PCA classification results for normalised and standardised DBpedia data.

Extremism

For the extremism dataset the best results were accomplished by normalising the data. Standardised and centered SVM is the only method that achieved a better F_1 value than its normalised counterpart. The best prediction, F_1 of 0.935, was delivered by kNN1 by normalising and standardising the data, whereas the worst prediction, F_1 of 0.691, results from Naïve Bayes by standardising and centering the data. Considering all algorithms, k-Nearest Neighbour with k=1 achieved the highest F_1 value with 0.957 for normalised and standardised PCA. The full results can be found in Table 4.19.

Alg.	Scale	Centered			Standardised		
		Precision	Recall	F_1	Precision	Recall	F_1
J48	Norm	0.923	0.923	0.923	0.935	0.935	0.935
	Std.	0.867	0.867	0.867	0.863	0.863	0.863
Naïve Bayes	Norm	0.702	0.701	0.701	0.705	0.701	0.703
	Std.	0.700	0.683	0.691	0.703	0.692	0.697
kNN1	Norm	0.956	0.956	0.956	0.957	0.957	0.957
	Std.	0.878	0.879	0.878	0.873	0.874	0.873
kNN3	Norm	0.905	0.905	0.905	0.901	0.901	0.901
	Std.	0.870	0.871	0.870	0.865	0.865	0.865
SVM	Norm	0.709	0.710	0.709	0.734	0.734	0.734
	Std.	0.731	0.731	0.731	0.731	0.731	0.731

Table 4.19: PCA classification results for normalised and standardised Extremism data.

Drug

The classification results for the Drug data are presented in Table 4.20. Over the whole experiment, the normalised data achieved the best F_1 values. The standardised data achieved on average F_1 values of only half the size than for the normalised setting. The standardised PCA approach accomplished slightly higher accuracy than the centered one. However, the best classification was a result of centered PCA on normalised data using kNN1 (0.993).

For the standardised and centered approach all classifiers predicted only

the positive class which resulted in large offsets between precision and recall.

Unfortunately, the the classification of the SVM on the standardised and centered PCA did not finish in time to be presented in Table 4.20. None of the previous Drug experiments showed good results for the standardised SVM approach and moreover, for all other standardised and centered classifications only the positive class was predicted, leading to models with no predictive power. Therefore it is likely that the missing results would not display any valuable predictions.

Alg.	Scale	Centered			Standardised		
		Precision	Recall	F_1	Precision	Recall	F_1
J48	Norm	0.982	0.982	0.982	0.984	0.984	0.984
	Std.	0.346	0.588	0.436	0.418	0.397	0.407
Naïve Bayes	Norm	0.855	0.849	0.852	0.853	0.853	0.853
	Std.	0.346	0.588	0.436	0.643	0.651	0.647
kNN1	Norm	0.993	0.993	0.993	0.99	0.99	0.990
	Std.	0.346	0.588	0.436	0.343	0.323	0.333
kNN3	Norm	0.972	0.972	0.972	0.966	0.966	0.966
	Std.	0.346	0.588	0.436	0.419	0.422	0.420
SVM	Norm	0.863	0.848	0.855	0.874	0.861	0.867
	Std.				0.674	0.613	0.642

Table 4.20: PCA classification results for normalised and standardised Drug data.

4.2 Discussion

Having gathered all outcomes of the experiments one can now analyse if the results confirm the predictions from the hypotheses made in Section 3.1.1 or if the experiments contradict the expectations. Subsequently the overall best results for each dataset will be discussed in connection with the data scaling, choice of algorithm and the structure of the data.

4.2.1 Hypothesis 1

n-gram features improve the classification accuracy of machine learning algorithms independently of the data domain.

The best results for both feature sets and all datasets are presented again in Table 4.21. For the DBpedia and Extremism data the 2-gram features improved the classification. The DBpedia data was slightly better classified using the full set of features on the standardised data. The picture is less clear for the Extremism data, where J48 and kNN3 gained marginally higher F1 values when using only the Posit features. Nonetheless, the peak F1 value was achieved by including the 2-gram ratios and applying kNN1. Overall, one can conclude that the experiments on the DBpedia and Extremism data support the hypothesis.

The Drug data, on the other hand, turned out to contradict the expectations. For this data the two feature sets have the same F_1 measure for almost all algorithms on the normalised data. One has to consider that the F_1 values were already so high that it is possible that a better classification can not be achieved with the current data considering the pre-classified data was labelled manually. The results for the standardised data were much worse and will not be considered here as they would not be used for classification purposes.

These results do not agree with the expectations stated before. As seen in Figure 3.4c, the correlation of 2-gram features with other features was smaller than for the other datasets which would imply that these values add more information to the classification. However, the opposite appeared to be true. The hypothesis tried to conclude that 2-gram ratios are meaningful across all used datasets, which is only partly true. What crucial factors influence the importance of 2-gram ratios remains unclear as the Extremism and Drug datasets contradict each other.

Data	Features	Scale	Alg.	F_1
DBpedia	Posit	Std.	SVM	0.551
	Posit + 2-gram	Std.	SVM	0.557
Extremism	Posit	Norm.	J48	0.959
	Posit + 2-gram	Norm.	kNN1	0.965
Drug	Posit	Norm.	kNN1	0.995
	Posit + 2-gram	Norm.	kNN1	0.995

Table 4.21: Overall classification results for Hypothesis 1.

4.2.2 Hypothesis 2

A feature created from low-frequency n-grams is more important than a feature of high-frequency n-grams.

To evaluate this statement the Posit features were combined with both the low frequency and the high frequency ratios to compare their classification results. For the DBpedia data the F_1 values clearly showed that the high frequency values contributed more efficiently to the classification than the low frequency feature did. For the normalised data, the classifications performed equally poorly according to the F-measure. For the standardised data, however, the separation is clear: the high frequency feature contained more information.
The Extremism and Drug data performed equally well on both feature sets independent of the algorithm or the type of scaling. The best results for each dataset are summarised in Table 4.22. When comparing the results

Data	Features	Scale	Alg.	F_1
DBpedia	Posit + low	Std.	J48	0.495
	Posit + high	Std.	SVM	0.555
Extremism	Posit + low	Norm.	kNN1	0.965
	Posit + high	Norm.	kNN1	0.965
Drug	Posit + low	Norm.	kNN1	0.995
	Posit + high	Norm.	kNN1	0.995

Table 4.22: Overall classification results for Hypothesis 2.

in this table with the overall results of Hypothesis 1 (see Table 4.21) one can notice that the DBpedia data was better classified using all 2-gram ratios, whereas the Extremism and Drug data did not profit from using all 2-gram ratios as either the low or high frequency feature was sufficient. As a smaller feature set is always preferred this is valuable information. Instead of using all three frequency ratios for the Extremism and Drug datasets it seems enough to use only one, which can improve the runtime of the classifier. For most of the settings the runtime of the classifiers does not exceed five minutes, nevertheless, for the largest dataset (DBpedia) the runtime of the SVM model increased to up to 14 hours.

4.2.3 Hypothesis 3

n-gram features alone are not sufficient for classification models to perform well.

This experiment was conducted in order to test how efficient the 2-gram features are on their own. It was expected that the information content of those features is not broad enough to receive a well working classifier, but this view was partly discounted. Table 4.23 shows the summarised results for the three datasets. It stands out that the DBpedia data only achieved a maximum F_1 value of 0.364 reinforcing the hypothesis. Contrary to that are the classifications of the Extremism and Drug data, which accomplished 0.885 and 0.937 respectively. Especially the Drug classifications are therefore only slightly less accurate than for the Posit features or any of the combinations with the 2-gram features. This surprising result shows how important 2-gram data is for these types of datasets. Considering the results of hypothesis 1 it seems confusing that the full feature set of Posit and 2-gram data did not improve the classification much, when only the 2-gram features form a solid base for classification models. One explanation for this could be that the 2-gram ratios contain a similar amount of information that simply leads to the same classifications. Therefore both feature sets do not add any knowledge to each other. It is also likely that the accuracies achieved are the highest possible, as the data gathered by the web-crawlers was classified manually which is prone to introduce a classification bias. Not all instances might be correctly labelled making it impossible to achieve a perfect classifier. Therefore, F-measures of 0.995 and 0.965 for the Drug and Extremism data respectively, could be close to the best feasible outcome that can be reached with the current data labels. Keeping in mind that the DBpedia data was not classified well it needs to be further investigated for what types of data 2-grams are meaningful.

Data	Scale	Alg.	F_1
DBpedia	Std.	SVM	0.364
Extremism	Norm.	kNN1	0.885
Drug	Norm.	kNN1	0.937

Table 4.23: Overall classification results for Hypothesis 3.

4.2.4 Hypothesis 4

Using a subset of Posit and n-gram features leads to better classification results.

This experiment was set up to evaluate if Weka included filter methods have the ability to find a subset of features that can outperform any of the previously tested subsets. The best features for all datasets were found using the OneR evaluator combined with the Ranker search. The best 80% of the features were used which resulted in a subset that includes most of the Posit features (see Table 4.24). The subset for the DBpedia data included the overall and high frequency features, whereas for the Extremism and Drug data all 2-gram features were kept. These subsets are supported by the results of the experiments for hypothesis 2, as for the DBpedia data the high frequency feature was more valuable than the low frequency ratio. The classification results for the DBpedia data could not be improved by the filter method. With an F_1 value 0.515 the classification was worse than for any of the other feature combinations evaluated before. The filter method did, however, succeed on the Extremism data. Also chosen by the OneR measure, kNN1 improved the F_1 value minimally to 0.968 compared to the previous best results of 0.965 for all other feature settings. The feature set achieving this classification excluded features involving interjections and particles, but included all 2-gram features. Even though the improvement is only a gain of 0.003, it shows that the set of features can be improved. Further experiments could be applied in order to extract the most valuable combination of features.

Data	Features	Scale	Alg.	F_1
DBpedia	1, 4, 5, 6, 13, 14, 16, 23, 24, 25, 26, 27, 28	Std.	SVM	0.515
Extremism	0, 1, 2, 3, 4, 5, 6, 7, 8, 11, 13, 14, 15, 16, 17, 18, 19, 21, 23 , 24, 25, 26, 27, 28, 29	Norm.	kNN1	0.968
Drug	0, 1, 2, 3, 4, 5, 6, 11, 16, 17, 19, 21, 25, 26, 27, 28, 29	Norm.	kNN1	0.994

Table 4.24: Overall classification results for Hypothesis 4.

The classification of the Drug data did not improve any previous results.

The best feature subset reached an F_1 value of 0.994 close to the overall best result of 0.995 that was achieved by the Posit and low or high frequency values. The subset contained only 14 of the 27 Posit features and all three 2-gram features showing that the set works efficiently even though a lot of data was excluded from the computations.
For Extremism and Drug data the successful techniques are similar. Both were classified most accurately using the normalised data and the filter method OneR combined with the Ranker search. The selected subsets have many elements in common, but no distinct pattern of important features could be determined.
The runtime of the SVM improved drastically for the feature subsets of this experiment as it decreased to up to four hours instead of 14 hours for hypothesis 1 and 2.

4.2.5 Hypothesis 5

Using Principal Component Analysis improves the effectiveness of the features and leads to better classification results.

Throughout this experiment this theory could not be confirmed. For all datasets the PCA based classifications were evaluated to be worse than any of the previous experiments.
DBpedia had a F_1 value of only 0.5, whereas the Posit and 2-gram features achieved a score of 0.557. The filter method achieved an F_1 value of 0.968, whereas PCA only reached 0.957. The best classification for the Drug data (0.995) was missed by just 0.002. The best results and their settings can be found in Table 4.25.

Data	PCA Type	Scale	Alg.	F_1
DBpedia	Centered	Std.	SVM	0.500
	Standardised	Std.	SVM	0.500
Extremism	Centered	Norm.	kNN3	0.905
	Standardised	Norm.	kNN1	0.957
Drug	Centered	Norm.	kNN1	0.993
	Standardised	Norm.	J49	0.984

Table 4.25: Overall classification results for Hypothesis 5.

4.2.6 Further Comments

When looking over all experiments, a pattern among the datasets becomes apparent. For all experiments the DBpedia data was classified best using SVM on standardised data. Only for the Posit + low frequency feature J48 served as a better classifier.
For hypothesis 1 to 4 the Extremism and Drug data were best classified with kNN1 on normalised data with only one exception which was the classification of the Extremism data using only Posit features, which was classified best using J48. The PCA experiment shows a slightly different outcome. Here the centered approach on Extremism data achieved the highest F_1 value with kNN3 and the best result of the Drug data was reached by applying J48 to the standardised PCA.
Throughout all experiments the Naïve Bayes classifier did not achieve higher precision and recall than any of the other classifiers. The Support Vector Machines showed good results on the DBpedia data, but not on the other datasets. J48, which is the implementation of a Decision Tree performed well on the Extremism and Drug data, as did kNN3. However, in several experiments kNN1 outperformed both.
The reason why k-Nearest Neighbour performs better with k=1 than k=3 might be due to the fact that for k=1 the model is close to the training data as it only includes the nearest data point in its calculations and has therefore a low bias. However, the probability to model noise in the data is high for the same reason. The predictions in the experiments might be excellent, because the testing data is similar to the training data chosen.

These differences between DBpedia and the other text corpora show not all text data can be classified using the same techniques. Nonetheless, it was also observed that the two datasets extracted from the surface and dark web show great resemblance when it comes to choosing a scaling method or a machine learning algorithm.
Another important aspect are the unbalanced predicted categories for DBpedia and Drug data. Choosing unsuitable scaling and classification models lead not only to low F_1 values but also to unbalanced precision and recall values. This gap between those two measures shows that when predicting the categories some were neglected completely or only predicted in a few instances. This effect did not influence the overall results of the experiments as they only occurred for the worst classifications. Nevertheless, this shows

how important both precision and recall are, as neither one of them alone is enough to judge the effectiveness of a classification model.
The developed subsets for Extremism and Drug data in hypothesis 4 showed that the features involving nouns were important to all evaluation techniques. Considering this, nouns seem to be valuable throughout the surface and dark web data which supports the approach of Weir et al. (2016) on using the sentiment of nouns for classifications.
In Section 3.2.2 the distribution of the number of sentences and words and of the average sentence length were shown in histograms. The distribution of the DBpedia data differed from the rest of the datasets as that data showed far less outliers. Even though the data was considered to include less noise and far more reasonable values for length of sentences, the classifications turned out to be less promising. With a F_1 value of 0.557 and 55.67% correctly classified instances this data corpus was the hardest to classify.
The other extreme was represented by the Drug data. The histograms showed many instances not agreeing on the sentence length with the rest of the dataset. Especially the average sentence length and the number of sentences led to the conclusion that the data contains a high level of noise. It was assumed that the text extracted from web pages includes parts not structured in sentences such as buttons. Those non-sentence texts were assumed to serve as a good basis for classification which seems to be correct as the Drug data achieved the highest F_1 value among all datasets (99.5%). 99.47% of all instances were classified correctly which is an astonishing result. The Extremism data also reached high accuracy with 96.80% correctly classified instances.
Another difference between the two types of data was presented in the heatmaps in Figure 3.4. Those maps showed the correlation between features. The DBpedia data has the most correlations between features, whereas the Drug data showed the least correlations. This might have influenced the classifications as well, as the features representing the DBpedia were more correlated and could not be used efficiently.

Overall, it has been shown that the Posit features were used efficiently on the Extremism and Drug data. The DBpedia data did not produce as good results, which does not necessarily mean that the Posit tool can not succeed on this type of data. It has to be mentioned that classifying the DBpedia

data was a harder task as there are not only more classes to predict, but the classes are also not related to each other. For the DBpedia data it had to be decided if an instance belonged to the category Company, Educational Institution, Artist or Athlete, whereas for the other datasets it had to be determined if a text was positive, negative or neutral towards extremism or positive or negative towards drugs. This means for Extremism and Drug data the categories evolved around only one objective: extremism or drugs, whereas classifying DBpedia data involved four distinct topics. This should make it harder to classify the DBpedia data and could be an explanation for the poor classification accuracies.
The Posit and 2-gram features achieved F_1 values of 0.968 and 0.995 for Extremism and Drug data respectively. Therefore one can conclude that those feature sets are very effective on the surface and dark web data considered for these experiments.

Chapter 5

Conclusion and Future Work

After conducting all experiments and discussing their results one can now conclude if the objectives of this dissertation were achieved. Moreover, several aspects can be pointed out that should be addressed by further research as they could not be covered in the range of this dissertation.

5.1 Conclusion

The main interest of this dissertation was to explore the effectiveness of the Posit and n-gram features on classification accuracy. As the Posit tool has only been applied to the Extremism data before (Weir et al., 2016) it was interesting to see how well the extracted features perform on other datasets. The experiments on the Drug data support the assumptions that Posit enriches the data and can be well appropriated for machine learning as the evaluation measures were higher than 0.99.
Having conducted several experiments on three datasets including 2-gram features it seems that those values have a limited impact on the overall classification. Combined with the Posit features a minimal improvement was achieved, however, on their own 2-gram features could develop a surprisingly effective classifier. Due to the Posit features offering a strong foundation for machine learning models, it was difficult to further improve those results. For the DBpedia data the classification results could be enhanced by the 2-gram features as for this data there was a wider range of possible improvement. It was shown that replacing some Posit features with 2-gram ratios can slightly increase the classification accuracy. The experiments showed that 2-gram ratios do not decrease the effectiveness of

classification models, but how much more accuracy can be added by using them depends on the dataset itself and on how valuable the Posit features are. If Posit features already develop a very well working classifier, the 2-gram features do not have a great impact on the outcome.
It is still believed that feature selection is a beneficial tool to increase the classifier's performance, however, the importance of individual features seems to be dependent on the dataset. For the Extremism and Drug data a certain pattern across all feature subsets evolved: the noun and noun type features were evaluated to be important for all subsets. In order to make stronger statements one has to conduct more tests on feature subsets as the decision to select only the features with at least 80% of the best rank resulted in subsets too small to offer good classifications.
An assumption that can be made concerning the scaling technique of the datasets is that the data collected by the web-crawlers showed better results throughout all experiments when normalising the data instead of standardising. This could be connected to the wide ranging distribution of several sentence features in the crawling data in contrast to the DBpedia texts. Moreover, the five machine learning models that were applied showed different levels of efficiency on the three datasets. For the DBpedia data Support Vector Machines were most adequate, whereas on the other web data they were outperformed by k-Nearest Neighbour which performed only slightly better than J48.

5.2 Future Work

For future work it could be useful to further explore the predictive power of n-gram ratios. The experiments conducted in this dissertation included only bi-grams, however, longer word sequences could be meaningful for classifications as they contain more context of the text and could therefore lead to even better classifications.
Furthermore, as the 2-gram ratios were more efficient on the Extremism and Drug corpora than on the Dbpedia data, it should be further explored on what types of data 2-gram frequencies are important. The experiments here do not fully conclude this as the DBpedia texts did not only differ in topic, but also in the number of classes. One could conduct the same experiments again with a different set-up of the DBpedia data. It would be

interesting to see if the classifications improve when using one topic as the positive class and a combination of the other topics as the negative class, e.g. Athlete versus non-Athlete texts. The suggestion is that these classes can achieve a higher classification score than the four categories used for this dissertation.

This leads us to another aspect that could be further explored. When discussing the distribution of the three datasets and their sentence related features, it emerged that the web-crawling data included many texts with abnormally long sentences. These texts were assumed to originate from non-sentence texts on the web pages (e.g. buttons). One element that could be implemented differently is the sentence count of the Posit tool. The original implementation only counted sentences ended by a full stop and at least one space. The variation used for the experiments also allowed sentences followed by a full stop but no space, which in turn counted URLs of web pages etc. as sentences. Further improvements could be made by restricting the sentence count to not accept URLs.
Another way of approaching this issue would be to deal with the button-type texts separately from the sentence-structured texts. This could be achieved by analysing the extracted HTML data instead of the raw texts. This would not only allow for new features, but also resolve the problem of unrealistically long sentences. Therefore, the Extremism and Drug features for average sentence length and number of sentences would include less noise, as those instances can be treated differently as button information.

Appendix A

Statistics

The following three tables show the statistics over the three datasets. Included are minimum, maximum, mean and standard deviation of each feature.

Feature	Min	Max	Mean	Std Dev.
Average sentence length	1.57	66894.00	105.54	664.68
Average word length	2.15	2434.71	8.21	20.10
Number of characters	27.00	944095.00	15344.84	32085.29
Number of sentences	1.00	11382.00	271.38	1064.49
Total unique words (types)	3.00	66895.00	616.76	1132.28
Total words (tokens)	3.00	149371.00	1908.46	3729.39
Type/Token Ratio	0.01	93.29	0.55	0.80
adjective types	0.00	182.00	10.27	12.81
adverb types	0.00	56.00	4.55	6.66
determiner types	0.00	15.00	3.26	2.52
interjection types	0.00	7.00	0.04	0.24
noun types	2.00	958.00	198.45	161.92
particle types	0.00	8.00	0.27	0.67
personal pronoun types	0.00	21.00	2.87	2.91
possessive pronoun types	0.00	21.00	2.87	2.91
preposition types	0.00	65.00	6.67	5.67
verb types	0.00	142.00	18.49	21.92
adjectives	0.00	418.00	44.86	44.64
adverbs	0.00	456.00	14.32	23.85
determiners	0.00	276.00	21.43	30.85
interjections	0.00	132.00	0.12	1.20
nouns	4.00	1948.00	707.94	554.06
particles	0.00	94.00	1.14	5.56
personal pronouns	0.00	286.00	13.45	24.80
possessive pronouns	0.00	286.00	13.45	24.80
prepositions	0.00	358.00	34.83	42.89
verbs	0.00	552.00	60.26	75.15
2-gram overall ratio	0.00	312.68	1.52	2.60
2-gram high freq ratio	0.00	903.85	1.64	6.69
2-gram low freq ratio	0.00	43.44	1.42	0.73

Table A.1: Drug Feature Statistics.

Feature	Min	Max	Mean	Std Dev.
Average sentence length	1.33	201.00	18.57	7.18
Average word length	4.17	202.20	6.30	0.78
Number of characters	10.00	1926.00	300.63	129.01
Number of sentences	1.00	28.00	2.82	1.49
Total unique words (types)	2.00	196.00	38.14	14.58
Total words (tokens)	2.00	308.00	48.01	20.85
Type/Token Ratio	0.27	2.00	0.83	0.11
adjective types	0.00	20.00	3.17	2.08
adverb types	0.00	12.00	0.91	1.10
determiner types	0.00	7.00	2.04	0.75
interjection types	0.00	1.00	0.00	0.01
noun types	1.00	150.00	18.67	7.64
particle types	0.00	3.00	0.04	0.20
personal pronoun types	0.00	7.00	0.75	0.83
possessive pronoun types	0.00	7.00	0.75	0.83
preposition types	0.00	15.00	3.72	1.90
verb types	0.00	34.00	5.15	2.63
adjectives	0.00	74.00	6.78	4.63
adverbs	0.00	28.00	1.87	2.29
determiners	0.00	50.00	8.19	4.95
interjections	0.00	4.00	0.00	0.03
nouns	2.00	344.00	43.27	18.93
particles	0.00	6.00	0.08	0.41
personal pronouns	0.00	28.00	1.98	2.55
possessive pronouns	0.00	28.00	1.98	2.55
prepositions	0.00	88.00	11.77	7.02
verbs	0.00	86.00	11.99	6.21
2-gram overall ratio	0.00	2.00	1.04	0.05
2-gram high freq ratio	0.00	3.00	1.04	0.05
2-gram low freq ratio	0.00	6.00	1.02	0.14

Table A.2: DBpedia Feature Statistics.

Feature	Min	Max	Mean	Std Dev.
Average sentence length	1.61	3612.00	46.35	66.63
Average word length	3.00	208.71	8.08	5.60
Number of characters	9.00	827255.00	13668.57	21936.16
Number of sentences	1.00	3356.00	57.96	96.51
Total unique words (types)	2.00	10600.00	664.26	622.62
Total words (tokens)	2.00	121324.00	1801.19	2880.69
Type/Token Ratio	0.01	1.50	0.48	0.14
adjective types	0.00	82.00	20.59	15.55
adverb types	0.00	51.00	7.14	7.29
determiner types	0.00	17.00	4.27	2.64
interjection types	0.00	4.00	0.08	0.29
noun types	2.00	606.00	266.68	137.40
particle types	0.00	9.00	0.76	1.19
personal pronoun types	0.00	22.00	4.49	4.22
possessive pronoun types	0.00	22.00	4.49	4.22
preposition types	0.00	49.00	12.07	6.07
verb types	0.00	140.00	39.54	29.54
adjectives	0.00	258.00	58.00	48.03
adverbs	0.00	178.00	19.87	22.57
determiners	0.00	334.00	53.51	50.03
interjections	0.00	318.00	0.36	3.29
nouns	4.00	1692.00	840.70	413.66
particles	0.00	98.00	2.26	4.63
personal pronouns	0.00	272.00	22.43	29.19
possessive pronouns	0.00	272.00	22.43	29.19
prepositions	0.00	320.00	92.11	61.26
verbs	0.00	926.00	122.67	111.38
2-gram overall ratio	0.00	14.93	1.31	0.33
2-gram high freq ratio	0.00	14.56	1.33	0.36
2-gram low freq ratio	0.00	16.09	1.25	0.33

Table A.3: Extremism Feature Statistics.

Appendix B

Implementation

B.1 Posit Changes

To move the results of the Posit tool for one text into a folder to gather all n-gram and summary data the following script was written:

```
#get name of data file
name=$(basename "$1" ".txt")
#create new file structure for summary data and n-grams
resultsFolder=/opt/posit/result
summary=$resultsFolder/summary_$name
ngram=$resultsFolder/ngram_$name

if [ -d "$summary" ] && [ "$2" = "--sum" ]; then
    rm -R $summary
fi
if [ -d "$ngram" ] && [ "$2" = "--ngram" ]; then
    rm -R $ngram
fi

oldSummaryFolder=$(pwd)/results
oldNgramFolder=$(pwd)/$name"_ngram_results"
#rename results folder to new name

#echo $parent $resultsFolder
if [ "$2" = "--sum" ]; then
    mv $oldSummaryFolder $summary
fi
if [ "$2" = "--ngram" ]; then
    mv $oldNgramFolder $ngram
fi
```

The method to count sentences had to be changed. Instead of at least one space also zero spaces are not allowed:

```
tr '\r\n' ' ' | sed 's/\([.?!]\)\s*\([A-Z]\)/\1\n\2/g' | awk
    '1'
```

The script to calculate the Posit summary data:

```
echo 'start posit'
pos_all.sh $1
echo 'rename folders'
rename_results.sh $1 $2
```

The script to calculate the n-gram frequencies:

```
echo 'start posit ngram'
ngram.sh $1
echo 'rename folders'
rename_results.sh $1 $2
```

B.2 Applying Posit

The main executions are handled with the following code:

main

```
#!/usr/bin/env python3
import sys
from constructor import construct
import reader.reader as rd

#DBPEDIA: skip: how many lines to skip, count: how many lines
    to analyse, id: unique run-id
#EXTREMIST: skip/count: not used, path: to index file, id:
    Category name (capitalised)
def run(data, path, skip, count, id, relation):
    if (data == "-drug"):
        crawlset = path.split('/')[-1]
        print(crawlset)
        #### creator for Drug data #####
        creator = construct(path, id, relation, ["crawlset",
            crawlset,"Positive, Negative, Unknown"])
        rd.read_files_drug(path, id, creator, skip, count)
```

```
    elif (data == "-extr"):
        #### creator for Extremism data #####
        creator = construct(path, id, relation, [None, None,"
            Positive, Negative, Neutral"])
        rd.read_files(path, id, creator)
    elif (data == "-dbp"):
        #### creator for DBpedia data #####
        creator = construct(path, id, relation, [None, None,"
            1,2,3,4"])
        rd.text_in_lines(skip, count, creator)


def index(data, path, skip, count):
    #### create the index files
    if (data == "-extr"):
        rd.read_index_extrem(skip, count, path)
    elif (data == "drug"):
        rd.read_index_drug(path)

run(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4], sys.
    argv[5], sys.argv[6])
```

reader

```
import csv
import re
from itertools import islice
from os.path import os, sys
from time import time
import zipfile
import chardet
import collections
import string
import langdetect

sys.path.append('..')
import arff.util as util
import arff.config as cfg

##============================================
## read the text from lines in a file
##============================================
def text_in_lines(skip, count, constructor):
```

```
# calculate the avg frequency for the n-gram ratios later
    on
#database = db.database()
ref_freq = 2052 #database.avg_freq()
skip = int(skip)
count = int(count)
with open(constructor.data_path, newline='') as file:
    attr = None
    counter = 0
    content = csv.reader(file, delimiter=',', quotechar='"'
        )
    for line in islice(content, skip, None):
        category = line[0]
        if int(category) not in [1,2,3,4]:
            util.write_log(constructor.log_path, ["wrong
                category, counter: %s" % (counter)])
            counter += 1
            continue
        text = line[2].strip()
        print(text[:200])
        if not isClean(text):
            util.write_log(constructor.log_path, ["text
                file mostly '?', counter: %s" % (counter)])
            counter += 1
            continue
        if sys.getsizeof(text) < 59:
            util.write_log(constructor.log_path, ["file too
                small, counter: %s" % (counter)])
            counter += 1
            continue
        constructor.read_text(text)

        #make sure results folder is empty
        try:
            print("REMOVE",cfg.results_path)
            os.remove(cfg.results_path)
        except OSError:
            pass

        attr, counter = constructor.execute(category, (skip
            +1), counter, attr, ref_freq)
        if counter == count:
            break
    constructor.finish()
```

```
##=============================================
## read the text from files in a folder
##=============================================
def read_files(index_path, category, constructor):
    ref_freq = 2052
    with open(index_path) as index:
        attr = None
        counter = 0
        content = index.readlines();
        for file in content:
            print(file[:-1])
            path = cfg.extremist_data + category + '.zip'

            with zipfile.ZipFile(path) as z:
                raw_text = z.read(file[:-1])
                print(raw_text[:30])
                #print(sys.getsizeof(raw_text))
                if sys.getsizeof(raw_text) < 59:
                    util.write_log(constructor.log_path, ["
                        counter: %s, file: %s, file too small" %
                         (counter, file[:-1])])
                    counter += 1
                    continue
                convert_encoding(raw_text)
                file_content = raw_text.decode("ISO-8859-1").
                    split('\n') #utf-8
                if len(file_content) > 1:
                    if file_content[1] == '':
                        util.write_log(constructor.log_path, ["
                            counter: %s, file: %s, empty second
                            line" % (counter, file[:-1])])
                        counter += 1
                        continue
                    text = file_content[1].split('\t')[1].strip
                        ()
                else:
                    util.write_log(constructor.log_path, ["
                        counter: %s, file: %s, empty file or
                        wrong encoding" % (counter, file[:-1])])
                    counter += 1
                    continue

                if not isClean(text):
```

```
                    util.write_log(constructor.log_path, ["
                        counter: %s, file: %s, text file mostly
                        '?'" % (counter, file[:-1])])
                    counter += 1
                    continue

                success = constructor.read_text(text)
                if success == -1 :
                    util.write_log(constructor.log_path, ["
                        counter: %s, file: %s, text file is too
                        large" % (counter, file[:-1])])
                    counter += 1
                    continue

                attr, counter = constructor.execute(category,
                    1, counter, attr, ref_freq)

        constructor.finish()


###============================================
## read the text from files in a folder
###============================================
def read_files_drug(index_path, category, constructor, skip =
    0, count = -1, additionalLine = False):
    ref_freq = 2052
    with open(index_path) as index:
        attr = None
        content = index.readlines();
        skip = int(skip)
        count = int(count)
        counter = skip+1
        if (count == -1):
            count = len(content)
        for i in range(skip,skip+count):
            if (i > len(content)-1):
                break
            line = content[i].rsplit(';',1)
            file = line[0]
            if isinstance(category,int):
                category = line[1].rstrip()
            print("Filename " +file)
            crawl = file.split('/')[0]
            path = cfg.drug_data + crawl + '.zip'
```

```
            with zipfile.ZipFile(path) as z:
                raw_text = z.read(file)
                print(raw_text[:40])
                #print(sys.getsizeof(raw_text))
                if sys.getsizeof(raw_text) < 59:
                    util.write_log(constructor.log_path, ["
                        counter: %s, file: %s, file too small" %
                         (counter, file[:-1])])
                    counter += 1
                    continue
                convert_encoding(raw_text)
                file_content = raw_text.decode("utf-8").replace
                    (u'\xa0', ' ')
                if additionalLine:
                    file_content = file_content.split('\n') #
                        utf-8ISO-8859-1
                    #print(file_content)
                    if len(file_content) > 1:
                        if file_content[1] == '':
                            util.write_log(constructor.log_path
                                , ["counter: %s, file: %s, empty
                                 second line" % (counter, file
                                [:-1])])
                            counter += 1
                            continue
                        text = file_content[1].split('\t')[1].
                            strip()
                    else:
                        util.write_log(constructor.log_path, ["
                            counter: %s, file: %s, empty file or
                             wrong encoding" % (counter, file
                            [:-1])])
                        counter += 1
                        continue
                else:
                    text = file_content
                try:
                    if not isClean(text):
                        util.write_log(constructor.log_path, ["
                            counter: %s, file: %s, text file
                            mostly '?'" % (counter, file[:-1])])
                        counter += 1
                        continue
```

```
                except:
                    util.write_log(constructor.log_path, ["
                        counter: %s, file: %s, encoding wrong" %
                         (counter, file[:-1])])
                    counter += 1
                    continue
                if (langdetect.detect(text) == 'en'):
                    try:
                        success = constructor.read_text(text)
                        if success == -1 :
                            util.write_log(constructor.log_path
                                , ["counter: %s, file: %s, text
                                file is too large" % (counter,
                                file[:-1])])
                            counter += 1
                            continue
                    except UnicodeEncodeError:
                        util.write_log(constructor.log_path, ["
                            counter: %s, file: %s, bad encoding"
                            % (counter, file[:-1])])
                        counter += 1
                        continue
                else:
                    util.write_log(constructor.log_path, ["
                        counter: %s, file: %s, wrong language" %
                        (counter, file[:-1])])
                    counter += 1
                    continue

                attr, counter = constructor.execute(category,
                    1, counter, attr, ref_freq)
                if (count != -1 & counter == count):
                    break
        constructor.finish()



#  create index for each folder, each line in index is name of
    file to be parsed by posit
#       line numbers represent ids
def read_index_extrem(skip, count, data_path):
    with zipfile.ZipFile(data_path) as z:
        print("unzipped")
        folder_dict = {}
```

```
        alpha_dic = {}
        for filename in z.namelist():
            if filename.endswith('/'):
                folder_dict[filename] = 0
                alpha_dic[filename] = list()
                file_path = cfg.data + 'ICCRC/index/' +
                    filename[:-1] + '.txt'

        for filename in z.namelist():
            if not filename.endswith('/'):
                for k in folder_dict.keys():
                    if k in filename:
                        folder_dict[k] += 1
                        file_path = cfg.data + 'ICCRC/index/' +
                            k[:-1] + '.txt'
                        alpha_dic[k].append(filename)

        with open(cfg.data + 'ICCRC/index/Temp-American,
            Americans,America.txt', 'w+') as tpc:
            for file in sorted(alpha_dic["Neutral/Temp-American
                ,Americans,America/"]):
                tpc.write(file+'\n')
        print(folder_dict)

def read_index_drug(data_path):
    numb_engl = 0
    numb_non_engl = 0
    pos_cat, neg_cat = readCategories()
    print(len(pos_cat),len(neg_cat))
    data_name = os.path.splitext(os.path.basename(data_path))
        [0]
    print(data_name)
    with zipfile.ZipFile(data_path) as z:
        print("unzipped")
        folder_dict = {}
        alpha_dic = {}
        for folder in z.namelist():
            if folder.endswith('/'):
                folder_dict[folder] = 0
                alpha_dic[folder] = list()
        for filename in z.namelist():
            if not filename.endswith('/'):
                raw_text = z.read(filename);
                raw_text = raw_text.decode("utf-8").replace(u'\
```

```
                    xa0', '␣')
                if (sys.getsizeof(raw_text) >= 59):
                    try:
                        if (langdetect.detect(raw_text) == 'en'
                            ):
                            numb_engl += 1
                            for k in folder_dict.keys():
                                if k in filename:
                                    folder_dict[k] += 1
                                    file_path = cfg.data + '
                                        drug/index' + k + '.txt'
                                    alpha_dic[k].append(
                                        filename)
                        else:
                            numb_non_engl += 1
                    except langdetect.lang_detect_exception.
                        LangDetectException:
                        print(raw_text)

        if not os.path.exists(cfg.data + 'drug/index'):
            os.makedirs(cfg.data + 'drug/index')
        with open(cfg.data + 'drug/index/'+data_name, 'w+') as
            tpc:
            for k in alpha_dic.keys():
                for file in sorted(alpha_dic[k]):
                    category = "Unknown"
                    domain_path = file.split('/')
                    if len(domain_path)>1:
                        domain = domain_path[1]
                    if domain in pos_cat:
                        category = "Positive"
                    elif domain in neg_cat:
                        category = "Negative"
                    tpc.write(file + ';' + category + '\n')
        print(numb_engl)
        print(numb_non_engl)

def convert_encoding(raw_text):
    result = chardet.detect(raw_text)
    charenc = result['encoding']
    print(charenc)

def isClean(text):
    common = collections.Counter(text).most_common(3)
```

```
    print(common)
    if(common[0][0] in string.punctuation):
        return False
    return True

def readCategories():
    drugs = set()
    non_drugs = set()
    with open(cfg.drug_data+"Drug_sites", 'r') as d:
        content = d.readlines()
        for site in content:
            site = site.replace("http://", "http_")
            site = site.replace('onion:80', 'onion_80')
            drugs.add(site.rstrip())
    with open(cfg.drug_data+"non_Drug_sites", 'r') as nd:
        content = nd.readlines()
        for site in content:
            site = site.replace("http://", "http_")
            site = site.replace('onion:80', 'onion_80')
            non_drugs.add(site.rstrip())
    return drugs, non_drugs
```

constructor

```
#!/usr/bin/env python3
from os.path import os, basename
import sys
import time
import config as cfg
import create_arff as read_index
import util

class construct:
    '''
    create the arff file
    set all tmp files in init
    '''
    def __init__(self, data, run_id, relation, data_id=None):
        self.start_time = time.time()
        self.data_path = data
        self.name = os.path.splitext(basename(data))[0] + str(
            run_id)
        self.relation = relation
```

```
        self.data_id = data_id
        self.tmp_path = cfg.posit_path +'/'+ self.name + '.txt'

        if not os.path.exists(cfg.new_results):
                os.makedirs(cfg.new_results)
        ## create temporary files
        self.attr_tmp_path = cfg.new_results + self.name + '
            _attr_tmp.txt'
        self.data_tmp_path = cfg.new_results + self.name + "
            _data_tmp.txt"
        self.arff_path = cfg.new_results + self.name + "
            _summary.arff"
        self.log_path = cfg.new_results + self.name + "_log"
        #make sure it's a new file, overwrite anything precious
        a = open(self.arff_path, "w")
        self.log = open(self.log_path, "w+")
        a.close()
        self.log.close()
        try:
            os.remove(self.attr_tmp_path)
        except OSError:
            try:
                os.remove(self.data_tmp_path)
            except OSError:
                pass
        self.data_tmp, self.attr_tmp = self.open_files()


    def execute(self, category, start, counter, attr, ref_freq)
        :
        #make sure results folder is empty
        try:
            print("REMOVE",cfg.results_path)
            os.remove(cfg.results_path)
        except OSError:
            pass
        #execute the summary script
        util.exec_shell('posit.sh '+ self.tmp_path + ' --sum',
            60)
        summary = cfg.new_results +'summary_'+ self.name + cfg.
            summary_path
        numeric, attributes = read_index.parse_summary([summary
            ])
        numeric[0].insert(0, str(start + counter))
```

```
        #execute the ngram script
        util.exec_shell('posit_ngram.sh '+ self.tmp_path + ' --
            ngram', 60)
        avg_ratios, high_ratios, low_ratios, ngram_attr=
            read_index.parse_ngram( 2, self.name, ref_freq)
        attributes = attributes + ngram_attr
        numeric[0] = numeric[0] + [str(avg_ratios[0]), str(
            high_ratios[0]), str(low_ratios[0])]
        print(numeric[0][-5])
        numeric[0].append(category)

        attr = util.write_attr(attr, attributes, self.
            attr_tmp_path, self.relation, self.data_id)
        util.dump(self.attr_tmp_path, self.arff_path)

        util.write_data(numeric[0], self.data_tmp, counter,
            self.data_id)

        counter += 1
        print("~~~~~~~~~~~~~~~~~~COUNTER: ",counter)

        if counter%4 == 0:
            util.dump(self.data_tmp_path, self.arff_path)
            self.data_tmp = open(self.data_tmp_path, "w")
        return attr, counter


    def open_files(self):
        data_tmp = open(self.data_tmp_path, "w+")
        attr_tmp = open(self.attr_tmp_path, "w+")
        print(data_tmp.write("@DATA" + os.linesep))
        util.write_log(self.log_path, [self.name, "Start time:
            " + str(self.start_time)])
        return data_tmp, attr_tmp

    '''
    Read the current text into a tmp file.
    returns -1 if file too large, 0 otherwise
    '''
    def read_text(self, text):
        with open(self.tmp_path, 'w+') as tmp:
            tmp.write(text)
            tmp.close()
            #print(text)
```

```
        fileSize = os.path.getsize(self.tmp_path)
        print('file size: ', fileSize)
        if fileSize > 1000000:
            return -1
        return 0

    def finish(self):
        util.dump(self.data_tmp_path, self.arff_path)
        # remove temporary files
        os.remove(self.attr_tmp_path)
        os.remove(self.data_tmp_path)
        os.remove(self.tmp_path)

        print("--- %s seconds ---" % (time.time() - self.
            start_time))
        util.write_log(self.log_path, ["Finish time: " + str(
            time.time), "Run time: %f seconds" % (time.time() -
            self.start_time)])
```

To create arff files this code was implemented:

create_arff

```
#!/usr/bin/env python3
import os
import re
import sys
import config as cfg
import util
sys.path.append('..')
import ngram.ngram_index as ngram
import ngram.ngram_ratio as ratio

def combine_dicts(summary_files):
    attributes = None
    data = list()
    for f in summary_files:
        [summary, token, pos] = util.parse(f)
        keys = [key for (key, value) in sorted(summary.items())
            ] + \
                [key for (key, value) in sorted(token.items())]
                    + \
```

```
                [key for (key, value) in sorted(pos.items())]
        values = [value for (key, value) in sorted(summary.
            items())] + \
                [value for (key, value) in sorted(token.items()
                    )] + \
                [value for (key, value) in sorted(pos.items())]
        ## get attributes from summary files and compare if
            they are equal, if not STOP
        if attributes is not None:
            assert attributes == keys
            if attributes != keys:
                print(f)
                raise ValueError("the attributes of the summary
                    files are not the same:", \
                                 attributes, keys)
        else:
            attributes = keys
        data.append(values)
    return attributes, data


# takes a set of summary files and creates a list of attributes
    and numerical data for each summary
def parse_summary(summary_files):
    print(summary_files)
    attributes, data = combine_dicts(summary_files)
    return data, attributes


# if ngram_file is None the methods searches for all ngram_*
    folders in the result folder
# if ngram_file is not None it only works on the specific file
def parse_ngram(n, ngram_file, ref_threshold):
    dir_list = os.listdir(cfg.new_results)
    files = []
    for i in range(0, len(dir_list)):
        if ngram_file is None:
            if ((not re.match('^.*\.(txt|arff)$', dir_list[i]))
                and \
                    re.match('^(ngram_).*$', dir_list[i])):
                files.append(cfg.new_results + dir_list[i] + "/
                    " + str(n) + "-grams.txt")
        else:
            if ((not re.match('^.*\.(txt|arff)$', dir_list[i]))
```

```
                and \
                    re.match('^(ngram_' + ngram_file + ').*$',
                        dir_list[i])):
                files.append(cfg.new_results + dir_list[i] + "/
                    " + str(n) + "-grams.txt")
    if ngram_file is not None:
        assert len(files) == 1
    # ratios for all files
    avg_ratios, high_ratios, low_ratios = [], [], []

    for f in files:
        print(f)
        n_gram = ngram.ngram(ref_threshold)
        try:
            n_gram.search_db(f)
            avg, high, low = ratio.avg_row_column(n_gram)
        except IndexError:
            avg, high, low = -1, -1, -1
            pass
        print("low_ratio:_", low)
        avg_ratios.append(avg)
        high_ratios.append(high)
        low_ratios.append(low)
    return avg_ratios, high_ratios, low_ratios, ['2-
        gram_overall_ratio', '2-gram_high_freq_ratio', '2-
        gram_low_freq_ratio']
```

util

```
#!/usr/bin/env python3
import os
import re
import subprocess

## parse summary.txt into string of attributes and data ##
def parse(posit):
    with open(posit) as f:
        content = f.readlines()
    content = [x.strip() for x in content]
    content.pop(0)
    attr_summary = {}
    attr_token = {}
    attr_pos = {}
```

```
    typ = 0
    for line in content:
        if (re.match('^\d+(\.\d+)?|inf[ \t:].+$', line)):
            arr = line.split(":")
            if(re.match('^[inf]$', arr[0].strip())):
                arr[0] = str(-1)
            if typ == 0:
                attr_summary[arr[1].strip().replace(" ", "_")]
                    = arr[0].strip()
            elif typ == 1:
                attr_token[arr[1].strip().replace(" ", "_")] =
                    arr[0].strip()
            elif typ == 2:
                attr_pos[arr[1].strip().replace(" ", "_")] =
                    arr[0].strip()
        if typ == 0 and line == "":
            if attr_summary.get("Number_of_sentences") == "0":
                attr_summary["Average_sentence_length_(ASL)"] =
                    "-1"
        elif typ == 0 and line == "NUMBER_OF_TOKEN_TYPES":
            typ = 1
        elif typ == 1 and line == "NUMBER_OF_POS_TYPES":
            typ = 2

    return [attr_summary, attr_token, attr_pos];

def dump(data_path, arff_file):
    arff = ""
    with open(data_path, "r") as data_tmp:
        for line in data_tmp:
            arff+=line

    with open(arff_file, "a") as a:
        a.write(arff)

# write attributes to temporary file
def write_attr(attr, attributes, attr_path, relation, data_id):
    with open(attr_path, 'w+') as attr_tmp:
        if attr is not None:
            if attributes != attr:
                raise ValueError("the attributes of the summary
                    files are not the same:", \
                                    attributes, attr)
        else:
```

```
            attr = attributes
            attr_tmp.write("@RELATION \t" + relation + os.
                linesep+ os.linesep)
            if data_id[0] != None:
                attr_tmp.write("@ATTRIBUTE \t" + data_id[0] + "
                    \t STRING" + os.linesep)
            attr_tmp.write("@ATTRIBUTE \t" + "ID" + "\t NUMERIC
                " + os.linesep)
            for a in attr:
                attr_tmp.write("@ATTRIBUTE \t" + a + "\t
                    NUMERIC" + os.linesep)
            attr_tmp.write("@ATTRIBUTE \tcategory\t {" +
                data_id[2] + "}" + os.linesep+ os.linesep)
    return attr

#write numerical data to temporary file
def write_data(data, data_tmp, counter, data_id):
    if data_id[1] != None:
        print(data_id[1])
        data.insert(0,data_id[1])
    for j in range(0,len(data)-1):
        data_tmp.write(data[j] + ",")
    data_tmp.write(data[-1] + os.linesep)
    data_tmp.truncate()

def write_log(log_file, text_list):
    with open(log_file, "a") as a:
        for t in text_list:
            a.write(t + os.linesep)

# command as string, time is timeout given before killing
    process
def exec_shell(command, time):
    Process=subprocess.Popen(command, shell=True)
    try:
        outs, errs = Process.communicate(timeout=time)
    except subprocess.TimeoutExpired:
            Process.kill()
            outs, errs = Process.communicate()
            return -1
    return 1
```

The n-gram ratios were calculated with the following code:

ngram_index

```
#!/usr/bin/env python3
import sys
import database_manager.database as db
from blinker._utilities import reference
sys.path.append('..')

# index the n-grams for each n in the file (case sensitive)
# seach blockwise in the reference files
class ngram:

    #grams is dictionary of ngrams in the current file
    def __init__(self, threshold):
        self.ref_threshold = threshold  #threshold of low vs
            high freq 2-grams
        self.grams = None       #2-gram in text with frequency
        self.ref_grams = {}     #2-gram from reference with
            freqency
        self.grand_total = 0    #overall freq of text + ref
        self.high_total = 0     #high freq of text + ref
        self.low_total = 0      #low freq of text + ref
        self.gram_high_col = 0   #text freq of high freq 2-
            grams
        self.gram_low_col = 0    #text freq of low freq 2-grams
        self.ref_high_col = 0   # only high freq overall
        self.ref_low_col = 0    # only low freq overall
        self.high_row = {}      # only row freq of high freq
        self.low_row = {}       # only row freq of low freq

    def update_col_totals(self, dic):#, db):
        for k,v in dic.items():
            if v != -1:
                gram_value = self.grams.get(k)
                #calculate high and low freq data
                if v >= self.ref_threshold:
                    self.ref_high_col += v
                    self.gram_high_col += gram_value
                    self.high_row[k] = v + gram_value
                elif v < self.ref_threshold:
                    self.ref_low_col += v
                    self.gram_low_col += gram_value
                    self.low_row[k] = v + gram_value
```

```
    def calc_high_totals(self):
        self.high_total = self.ref_high_col + self.
            gram_high_col

    def calc_low_totals(self):
        self.low_total = self.ref_low_col + self.gram_low_col

    def calc_grand_totals(self):
        self.grand_total = self.high_total + self.low_total

    def search_db(self,file):
        #ngrams =
        self.parse_ngrams(file)
        database = db.database()
        ref = "twogram.db"
        reference_freq = database.get_list_data(ref, self.grams
            .keys())
        self.print_ref(reference_freq)
        self.update_col_totals(reference_freq)
        self.calc_high_totals()
        self.calc_low_totals()
        self.calc_grand_totals() # call after high/low total
        self.ref_grams = reference_freq

    def parse_ngrams(self, file):
        with open(file) as gram:
            lines = gram.readlines()
        lines = [x.strip() for x in lines]
        dic = {}
        for l in lines:
            line = l.split(" ",1)
            ngram = line[1].replace("\t", " ")
            dic[ngram] = int(line[0])
        self.grams = dic

    def print_ref(self, reference):
        l = list()
        for k,v in reference.items():
            if v != -1:
                l.append(k)
        print(l)
```

ngram_ratio

```
#!/usr/bin/env python3

## Create pos-n-grams of text file
## calculate number of (occurences of n-gram / total number of
    n-grams (>1))
def row_column_ratio(ngram, n):
    total = ngram.high_row.copy()
    total.update(ngram.low_row)
    assert len(total) == (len(ngram.high_row) + len(ngram.
        low_row))
    row = total.get(n)
    if row is None:
        return -1
    col = ngram.gram_high_col + ngram.gram_low_col
    total = ngram.grand_total
    return (row*col)/total

def high_row_column_ratio(ngram, n):
    row = ngram.high_row.get(n)
    if row is None:
        return -1
    col = ngram.gram_high_col
    total = ngram.high_total
    return (row*col)/total

def low_row_column_ratio(ngram, n):
    row = ngram.low_row.get(n)
    if row is None:
        return -1
    col = ngram.gram_low_col
    total = ngram.low_total
    return float((row*col)/float(total))

def avg_row_column(ngram):
    rts, high_rts, low_rts = [], [], []
    for n in ngram.grams:
        #calculate row_column ratio for each ngram
        rts.append(row_column_ratio(ngram, n))
        high_rts.append(high_row_column_ratio(ngram, n))
        low_rts.append(low_row_column_ratio(ngram, n))

    return calc_avg(rts), calc_avg(high_rts), calc_avg(low_rts)
```

```
#calculate average of row_column ratios
def calc_avg(lisst):
    length = 0
    summ = 0
    for i in lisst:
        if i != -1:
            length += 1
            summ += i
    if length == 0:
        return 0
    return (summ / float(length))
```

In order to handle the database management this code was implemented:

Database

```
#!/usr/bin/env python3
import gzip
import os
from os.path import basename
from pathlib import Path
import re
import sys
sys.path.append('..')
import arff.config as cfg
import sqlite3 as lite

class database:

    ## one databaser per index file
    ## path defines the subset of .gz folders
    def create_db(self, folder):
      path = cfg.n_gram_db + "/" + folder
      for f in os.listdir(path):
        if not re.match(f, "2gm.idx"):
          name = os.path.splitext(f)[0]
          print(name)
          db_name = cfg.data + "/database/" + self.scrub(name)
              + ".db"
          if not Path(db_name).is_file():
            ngram_name = cfg.n_gram_db + "/" + folder +"/"+
                name + ".gz"
```

```
            db = lite.connect(db_name)

            with db:
              cur = db.cursor()
              command = "CREATE TABLE "+ self.scrub(name) +" (
                  ngram TEXT, frequency INT)"
              print(command)
              cur.execute(command)
              with gzip.open(ngram_name,'r') as ngrams:
                for line in ngrams:
                  line = line.decode("utf-8").strip()
                  arr = line.split('\t')
                  if arr[0].replace(" ", "").isalnum():
                    try:
                      command = "INSERT INTO "+ self.scrub(name
                          ) +" VALUES(\'"+arr[0]+"\',"+arr[1]+"
                          )"
                      cur.execute(command)
                    except IndexError:
                      print(arr, line)

    def open_db(self, db):
        conn = lite.connect(db)
        # Let rows returned be of dict/tuple type
        conn.row_factory = lite.Row
        print ("Openned database %s as %r" % (db, conn))
        return conn

    # one database for all 2-gram frequency data
    def complete_db(self, folder):
      path = cfg.n_gram_db + "/" + folder
      db_name = cfg.n_gram_db + "/" + folder + ".db"
      print(db_name)
      if not Path(db_name).is_file():
        db = lite.connect(db_name)
        with db:
          cur = db.cursor()
          command = "CREATE TABLE "+ "twograms" +" (ngram TEXT
              PRIMARY KEY, frequency INT)"
          print(command)
          cur.execute(command)
          for f in os.listdir(path):
            if not re.match(f, "2gm.idx"):
              print(f)
```

```
            ngram_name = cfg.n_gram_db + "/" + folder +"/"+ f
            with gzip.open(ngram_name,'r') as ngrams:
              for line in ngrams:
                line = line.decode("utf-8").strip()
                arr = line.split('\t')
                if arr[0].replace(" ", "").isalnum():
                  try:
                    command = "INSERT INTO "+ "twograms" +"
                        VALUES(?, ?)"
                    cur.execute(command, [arr[0], arr[1]])
                  except IndexError:
                    print(arr, line)


# Returns frequency if pattern was found in database, -1
    otherwise
def get_list_data(self, db, patterns):
    db = os.path.splitext(db)[0]
    con = lite.connect( cfg.n_gram_db + "/"+db+".db")
    #con = lite.connect( cfg.data + db +".db")
    print( cfg.data + db +".db")
   # print(list(patterns)[:50])
    dic = {k:-1 for k in patterns}
    pat = list(patterns)
    with con:
        cur = con.cursor()
        placeholder= '?'
        limit = 80
        while(len(pat) > 0):
            var_list = pat[0:limit]
            pat = pat[limit:]

            placeholders= ', '.join(placeholder for unused
                in list(var_list))
            command = "SELECT * FROM "+db+ " WHERE (ngram
                IN (%s));" % placeholders
            #print(command)
            cur.execute(command, list(var_list))
            rows = cur.fetchall()
            for row in rows:
                dic[row[0]] = row[1]

        return dic
```

```
    def avg_freq(self):
        print("Calculate avg freqencies of database")
        avg = 0
        #db_path = cfg.n_gram_db + "twogram.db"
        db_path = cfg.data + "twogram.db"
        con = lite.connect(db_path)
        with con:
            cur = con.cursor()
            command = "SELECT AVG(frequency) FROM "+ os.path.
                splitext(basename(db_path))[0] +";"
            print(command)
            cur.execute(command)
            avg = cur.fetchone()[0]
        print("threshold: ", avg)
        return avg

# only keep alphanumerics in name, starting with letter
def scrub(table_name):
    return ''.join( chr for chr in table_name if chr.isalnum()
        )[1:]
```

B.3 Applying Weka

Execute

```
package classify;

public class Execute {

 public static void main(String[] args) {
  String delete = null;
  String type = null;
  if (args.length<4)
   delete = "";
  else if (args.length>=4) {
   type = args[3];
   System.out.println(type);
   if  (args.length > 4)
    delete = args[4];
   else
    delete = "";
  }
```

```
  System.out.println(args[0] +" "+ args[1] +" "+ args[2] +" "+
      delete);
  String train = args[0];
  String test = args[1];
  boolean centered = false;
  if (args[2].equals("-C")) {
   centered = true;
  } else if (args[2].equals("-S")) {
   centered = false;
  } else {
   System.err.println("wrong input, "+args[1]+" must be either
       -C for centered or -S for standardised");
  }

  Driver run;
  try {
   if (delete.equals("") && type == null)
    run = new Driver(train, test, centered);
   else
    run = new Driver(train, test, centered, delete, type);
  } catch (Exception e) {
   e.printStackTrace();
  }
 }
}
```

Driver

```
package classify;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
```

```
import attributeSelection.SelectAttributes;
import utils.PrepareData;
import utils.Utils;
import weka.core.Instances;

public class Driver {

 private String fileName;
 private File f;
 private Path pcaFile;
 private String pcaFolder;
 private Path filterFile;
 private String filterFolder;
 private Path positNgramFile;
 private String folder;
 private Path ngramFile;
 private String ngramFolder;
 private String content;
 private Instances data;
 private Instances train;
 private Instances test;
 private String projectPath = "./";

 public Driver(String trainData, String testData, boolean
     centered) throws Exception {
  /** Standardise data, remove ID attribute and make category
      attribute nominal */
  train = PrepareData.createInstances(trainData);
  test = PrepareData.createInstances(testData);
  train = PrepareData.cleanData(train, true, false, "");
  test = PrepareData.cleanData(test, true, false, "");

  String dataset = Utils.getFileName(trainData);
  createFilesFilter(dataset, centered, "selectAttribute");
  List<Instances> train_test = new ArrayList<Instances>(Arrays.
      asList(train, test));

  /** PCA */
  Files.write(pcaFile, (trainData + (centered ? "\ncentered_
      data\n" : "\nstandardised_data\n")).getBytes(),
    StandardOpenOption.APPEND);
  executePCA(train_test, centered);

  /** FILTER */
```

```
 Files.write(filterFile, (trainData + (centered ? "\ncentered␣
     data\n" : "\nstandardised␣data\n")).getBytes(),
   StandardOpenOption.APPEND);
 executeFilter(train_test);

}

public Driver(String trainData, String testData, boolean
    centered, String delete, String type) throws Exception {

 /** Standardise data, remove ID attribute and make category
     attribute nominal */
 train = PrepareData.createInstances(trainData);
 test = PrepareData.createInstances(testData);
 train = PrepareData.cleanData(train, true, false, delete);
 test = PrepareData.cleanData(test, true, false, delete);

 String dataset = Utils.getFileName(trainData);

 Path path = null;
 if (type.equals("posit") || type.equals("ngram")) {
  createFilesPositNgram(dataset, type);
  path = positNgramFile;
 }

 /** Train the classifier and evaluate */
 RunClassifier.runClassifier(train, test, RunClassifier.
     buildJ48("-C␣0.25␣-M␣2", path),
   path);
 RunClassifier.runClassifier(train, test, RunClassifier.
     buildNaiveBayes("-K", path),
   path);
 RunClassifier.runClassifier(train, test, RunClassifier.
     buildKNN("", path, 1), path);
 RunClassifier.runClassifier(train, test, RunClassifier.
     buildKNN("", path, 3), path);
 RunClassifier.runClassifier(train, test, RunClassifier.
     buildSVN("", path), path);
}


private void executePCA(List<Instances> train_testSet, boolean
     centered) throws Exception {
 List<Instances> train_test_transformed = SelectAttributes.pca
```

```
        (train_testSet, pcaFile, centered);

    RunClassifier.runClassifier(train_test_transformed.get(0),
        train_test_transformed.get(1),
      RunClassifier.buildJ48("-C 0.25 -M 2", pcaFile), pcaFile);
    RunClassifier.runClassifier(train_test_transformed.get(0),
        train_test_transformed.get(1),
      RunClassifier.buildNaiveBayes("-K", pcaFile), pcaFile);
    RunClassifier.runClassifier(train_test_transformed.get(0),
        train_test_transformed.get(1),
      RunClassifier.buildKNN("", pcaFile, 1), pcaFile);
    RunClassifier.runClassifier(train_test_transformed.get(0),
        train_test_transformed.get(1),
      RunClassifier.buildKNN("", pcaFile, 3), pcaFile);
    RunClassifier.runClassifier(train_test_transformed.get(0),
        train_test_transformed.get(1),
      RunClassifier.buildSVN("", pcaFile), pcaFile);

  }

  private void executeFilter(List<Instances> train_test) throws
      Exception {
   Map<String, int[]> attrKept = SelectAttributes
     .filterAttributes(new ArrayList<Instances>(Arrays.asList(
         train_test.get(0))), filterFile, filterFolder);

   /** for each attribute filter method evaluate the subset on a
       J48 */
   for (Map.Entry<String, int[]> entry : attrKept.entrySet()) {
    this.content = "\n=======\n" + entry.getKey() + "\n=======\n";
    System.out.println(content);
    Files.write(filterFile, (content + "\n").getBytes(),
        StandardOpenOption.APPEND);

    /** Remove all attributes but the subset from the data */
    this.content = "\nSubset attributes: ";
    for (int j : entry.getValue()) {
     content += String.valueOf(j) + " " + this.train.attribute(j
         ).name() + ", ";
    }
    System.out.println(this.content);
    int[] toDelete = Arrays.copyOf(entry.getValue(), entry.
        getValue().length + 1);
    toDelete[toDelete.length - 1] = train_test.get(0).classIndex
```

```
        ();

    // invert the selection of attributes
    Files.write(filterFile, (this.content + "\n").getBytes(),
        StandardOpenOption.APPEND);
    Instances new_train = PrepareData.remove(train_test.get(0),
        toDelete, true);
    Instances new_test = PrepareData.remove(train_test.get(1),
        toDelete, true);

    /** Train the classifier and evaluate */
    RunClassifier.runClassifier(new_train, new_test,
        RunClassifier.buildJ48("-C 0.25 -M 2", filterFile),
      filterFile);
    RunClassifier.runClassifier(new_train, new_test,
        RunClassifier.buildNaiveBayes("-K", filterFile),
      filterFile);
    RunClassifier.runClassifier(new_train, new_test,
        RunClassifier.buildKNN("", filterFile, 1), filterFile);
    RunClassifier.runClassifier(new_train, new_test,
        RunClassifier.buildKNN("", filterFile, 3), filterFile);
    RunClassifier.runClassifier(new_train, new_test,
        RunClassifier.buildSVN("", filterFile), filterFile);

  }
}

private void emptyFile(File f) {
  if (f.exists()) {
    RandomAccessFile raf;
    try {
      raf = new RandomAccessFile(f, "rw");
      raf.setLength(0);
      raf.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
}

private void createFilesFilter(String dataset, boolean
    centered, String type) throws IOException {
```

```
  String c = (centered ? "_centered" : "_standardised");
  pcaFolder = this.projectPath + type + "/" + dataset + "/pca/"
      ;
  filterFolder = this.projectPath + type + "/" + dataset + "/
      filter/";
  ArrayList<String> folders = new ArrayList<String>(Arrays.
      asList(this.projectPath + type + "/",
    this.projectPath + type + "/" + dataset, pcaFolder,
        filterFolder));
  for (String folder : folders) {
   File file = new File(folder);
   if (!file.exists()) {
    file.mkdirs();
   }
  }
  fileName = pcaFolder + c + ".txt";
  f = new File(fileName);
  emptyFile(f);
  f.createNewFile();
  pcaFile = Paths.get(fileName);
  fileName = filterFolder + c + ".txt";
  f = new File(fileName);
  emptyFile(f);
  f.createNewFile();
  filterFile = Paths.get(fileName);
 }

 private void createFilesPositNgram(String dataset, String type
     ) throws IOException {
  folder = this.projectPath + type + "/" + dataset;
  File f = new File(folder);
  if (!f.exists())
   f.mkdirs();
  fileName = folder + ".txt";
  f = new File(fileName);
  emptyFile(f);
  f.createNewFile();
  positNgramFile = Paths.get(fileName);
 }
}
```

PrepareData

```
package utils;

import java.util.Arrays;

import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Center;
import weka.filters.unsupervised.attribute.MathExpression;
import weka.filters.unsupervised.attribute.Normalize;
import weka.filters.unsupervised.attribute.NumericToNominal;
import weka.filters.unsupervised.attribute.Remove;
import weka.filters.unsupervised.attribute.Standardize;
import weka.filters.unsupervised.instance.RemoveDuplicates;

public class PrepareData {

 public static Instances createInstances(String args) throws
    Exception {
  DataSource source = new DataSource(args);
  Instances data = source.getDataSet();
  if (data.classIndex() == -1)
   data.setClassIndex(data.numAttributes() - 1);
  return data;
 }

 public static Instances cleanData(Instances data, boolean
    toNominal, boolean removeDup, String toDelete) throws
    Exception {
  if (toNominal) {
   /** change category attribute to nominal */
   String[] options = weka.core.Utils.splitOptions("-R last");
   NumericToNominal convert = new NumericToNominal();
   convert.setOptions(options);
   convert.setInputFormat(data);
   data = Filter.useFilter(data, convert);
  }

  if (toDelete.length() > 0) {
   assert !data.isEmpty();
   String optionsStr = "-R " + toDelete;
   System.out.println(optionsStr);
   String[] options_remove = weka.core.Utils.splitOptions(
```

```
      optionsStr);

   Remove remove = new Remove();
   remove.setOptions(options_remove);
   remove.setInputFormat(data);
   data = Filter.useFilter(data, remove);
  }
  if (removeDup) {
   RemoveDuplicates dup = new RemoveDuplicates();
   dup.setInputFormat(data);
   data = Filter.useFilter(data, dup);
  }

  return data;
 }

 public static Instances standardise(Instances data) throws
     Exception {
  Standardize standard = new Standardize();
  standard.setInputFormat(data);
  return Filter.useFilter(data, standard);
 }

 public static Instances normalise(Instances data) throws
     Exception {
  Normalize norm = new Normalize();
  norm.setInputFormat(data);
  return Filter.useFilter(data, norm);
 }

 public static Instances center(Instances data) throws
     Exception {
  Center centerFilter = new Center();
  centerFilter.setInputFormat(data);
  return Filter.useFilter(data, centerFilter);
 }

 public static Instances multiply(Instances data, String m)
     throws Exception {
  MathExpression multiply = new MathExpression();
  multiply.setExpression("A * " + m);
  multiply.setInputFormat(data);
  return Filter.useFilter(data, multiply);
 }
```

```
 public static Instances remove(Instances data, int[] rem,
    boolean inverse) throws Exception {
  int[] toDelete = rem.clone();
  for (int i=0; i<toDelete.length; i++) {
   toDelete[i] += 1;
  }
  String delete = "-R ";
  if (inverse)
   delete = "-V " + delete;
  String d = Arrays.toString(toDelete).replace(" ","");
  delete += d.substring(1, d.length()-1);
  System.out.println(delete);
  String[] options_remove = weka.core.Utils.splitOptions(delete
      );

  Remove remove = new Remove();
  remove.setOptions(options_remove);
  remove.setInputFormat(data);
  data = Filter.useFilter(data, remove);

  if (data.classIndex() == -1) {
   data.setClassIndex(data.numAttributes()-1);
  }
  return data;
 }

}
```

SelectAttributes

```
package attributeSelection;

import java.io.File;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
```

```
import java.util.List;
import java.util.Map;

import graph.RankingGraph;
import utils.AttributeUtil;
import utils.DataSplit;
import utils.Pair;
import utils.PrepareData;
import weka.attributeSelection.ASEvaluation;
import weka.attributeSelection.ASSearch;
import weka.attributeSelection.AttributeSelection;
import weka.attributeSelection.BestFirst;
import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.CorrelationAttributeEval;
import weka.attributeSelection.GainRatioAttributeEval;
import weka.attributeSelection.GreedyStepwise;
import weka.attributeSelection.InfoGainAttributeEval;
import weka.attributeSelection.OneRAttributeEval;
import weka.attributeSelection.PrincipalComponents;
import weka.attributeSelection.Ranker;
import weka.attributeSelection.WrapperSubsetEval;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.trees.J48;
import weka.core.Instances;
import weka.core.SelectedTag;
import weka.core.Utils;

public class SelectAttributes {

 public static Map<String, int[]> filterAttributes(Collection<
    Instances> trainingSets, Path file, String filterFolder)
    throws Exception {
  Map<String, int[]> attrSubset = new HashMap<>();
  AttributeSelection attsel = new AttributeSelection();
  List<Object> evalMethods = new ArrayList<>();
  List<Object> searchMethods = new ArrayList<>();
  /** create list of evaluation and search methods */
  OneRAttributeEval oneREval = new OneRAttributeEval();
  evalMethods.add(oneREval);
  CfsSubsetEval cfsEval = new CfsSubsetEval();
  evalMethods.add(cfsEval);
  CorrelationAttributeEval corEval = new
     CorrelationAttributeEval();
  evalMethods.add(corEval);
```

```
GainRatioAttributeEval gainRatioEval = new
    GainRatioAttributeEval();
evalMethods.add(gainRatioEval);
InfoGainAttributeEval gainInfoEval = new
    InfoGainAttributeEval();
evalMethods.add(gainInfoEval);

GreedyStepwise greedySearch = new GreedyStepwise();
searchMethods.add(greedySearch);
Ranker rankSearch = new Ranker();
searchMethods.add(rankSearch);
BestFirst bestFirstSearch = new BestFirst();
searchMethods.add(bestFirstSearch);

Map<String, List<Pair<String, Double>>> methodAvg = new
    HashMap<>();
for (Object eval : evalMethods) {
 for (Object search : searchMethods) {
  if (((eval instanceof CfsSubsetEval || eval instanceof
      WrapperSubsetEval) && search instanceof Ranker)
     || ((eval instanceof CorrelationAttributeEval || eval
        instanceof GainRatioAttributeEval
       || eval instanceof InfoGainAttributeEval || eval
          instanceof OneRAttributeEval)
      && !(search instanceof Ranker))) {
   continue;
  }
  long startTime = System.currentTimeMillis();
  String[] evalName = eval.toString().split(" ");
  String[] searchName = search.toString().split(" ");

  String method = (evalName[0].trim() + "_" + evalName[1].
      trim() + searchName[0].trim() + "_"
    + searchName[1].trim()).replace(".", "").replaceAll("\\s+
        ", "");
  Files.write(file, ("\n==========\
      nmethod: " + method + "\n").getBytes(),
    StandardOpenOption.APPEND);

  if (search instanceof GreedyStepwise) {
   ((GreedyStepwise) search).setSearchBackwards(true);
  } else if (search instanceof BestFirst) {
   ((BestFirst) search).setOptions(weka.core.Utils.
       splitOptions("-D 2 -N 5"));
```

```
    }
    attsel = new AttributeSelection();
    attsel.setEvaluator((ASEvaluation) eval);
    attsel.setSearch((ASSearch) search);
    attsel.setRanking(true);
    Iterator<Instances> it = trainingSets.iterator();
    for (int i = 0; i < trainingSets.size(); i++) {
     Files.write(file, ("========= Training set number: " +
         String.valueOf(i) + "\n").getBytes(),
       StandardOpenOption.APPEND);
     Instances train = (Instances) it.next();
     if (eval instanceof GainRatioAttributeEval || eval
         instanceof InfoGainAttributeEval
       || eval instanceof CfsSubsetEval)
      train = PrepareData.multiply(train, "pow(10,10)");

     try {
      attsel.SelectAttributes(train);
     } catch (IllegalArgumentException e) {
      System.out.println(e.getMessage());
      Files.write(file, (e.getMessage() + "\n").getBytes(),
          StandardOpenOption.APPEND);
      continue;
     }

     // obtain the attribute indices that were selected
     System.out.println( attsel.toResultsString());
     Files.write(file, (attsel.toResultsString() + "\n").
         getBytes(),
       StandardOpenOption.APPEND);

     int[] indices = attsel.selectedAttributes();
     System.out.println(Utils.arrayToString(indices));
     Files.write(file, (Utils.arrayToString(indices) + "\n").
         getBytes(), StandardOpenOption.APPEND);

     if (indices.length < train.numAttributes()) {
      List<String> names = new ArrayList<>();
      for (int j : indices) {
       String attrName = train.attribute(j).name();
       names.add(attrName);
      }
      attrSubset.put(method, indices);
     }
```

```
    try {
     double[][] ranking = attsel.rankedAttributes();
     List<Pair<String, Double>> rankings = new ArrayList<>();
     Pair<String, Double> rank = new Pair<String, Double>();
     double threshold = ranking[0][1] * (double) 0.8;
     List<Integer> keptAttr = new ArrayList<>();
     for (double[] r : ranking) {
      if (r[1] > threshold) {
       keptAttr.add((int) r[0]);
      }
      String attrName = train.attribute((int) r[0]).name();
      String weight = r[1] + "\t\t" + attrName;
      System.out.println(weight);
      rank = new Pair<String, Double>(attrName, r[1]);
      rankings.add(rank);
     }
     int[] tmp = new int[keptAttr.size()];
     for (int j = 0; j < keptAttr.size(); j++)
      tmp[j] = keptAttr.get(j);
     attrSubset.put(method, tmp);
    } catch (Exception e) {
     continue;
    }
    Files.write(file, (String.valueOf(System.currentTimeMillis
        () - startTime) + "\n").getBytes(),
      StandardOpenOption.APPEND);
   }
  }
 }
 return attrSubset;
}

public static List<Instances> pca(List<Instances>
    train_testSet, Path pcaFile, boolean centered) throws
    Exception {
 String content = "\n";
 Instances train = train_testSet.get(0);
 Instances test = train_testSet.get(1);

 PrincipalComponents pca = new PrincipalComponents();
 pca.setCenterData(centered);
 pca.setMaximumAttributeNames(30);
 pca.buildEvaluator(train);
```

```
    content = String.valueOf(pca.getVarianceCovered());
    Files.write(pcaFile, (content).getBytes(), StandardOpenOption
        .APPEND);
    Instances train_trans = pca.transformedData(train);
    Instances test_trans = pca.transformedData(test);

    content = "\ntransformed attributes\n";
    for (int j = 0; j < train_trans.numAttributes(); j++) {
     content += pca.evaluateAttribute(j) + " " + j + " " +
         train_trans.attribute(j).name() + "\n";
    }
    Files.write(pcaFile, (content).getBytes(), StandardOpenOption
        .APPEND);

    content = "\n\n==========\n" + pca.toString();
    Files.write(pcaFile, (content).getBytes(), StandardOpenOption
        .APPEND);

    return new ArrayList<Instances>(Arrays.asList(train_trans,
        test_trans));
  }
}
```

RunClassifier

```
package classify;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardOpenOption;
import java.util.Arrays;
import java.util.stream.Collectors;

import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.classifiers.functions.SMO;
import weka.classifiers.Classifier;
import weka.classifiers.lazy.IBk;
import weka.classifiers.trees.J48;
import weka.core.Instances;
import weka.core.SelectedTag;
```

```
public class RunClassifier {
 private static String content;

 public static void runClassifier(Instances train, Instances
    test, Classifier clf, Path file) throws Exception {
  long startTime = System.currentTimeMillis();
  clf.buildClassifier(train);

  Evaluation eval = new Evaluation(train);
  eval.evaluateModel(clf, test);

  content = eval.toSummaryString("\nResults\n======\n", true);
  System.out.println(content);
  Files.write(file, content.getBytes(), StandardOpenOption.
     APPEND);

  content = eval.toClassDetailsString();
  System.out.println(content);
  Files.write(file, content.getBytes(), StandardOpenOption.
     APPEND);

  content = eval.toMatrixString();
  System.out.println(content);
  Files.write(file, content.getBytes(), StandardOpenOption.
     APPEND);

  double roc = eval.areaUnderROC(0);
  content = "\n Area Under ROC: \n" + String.valueOf(roc);
  Files.write(file, (content+"\n").getBytes(),
     StandardOpenOption.APPEND);

  long endTime = System.currentTimeMillis();
  long totalTime = endTime - startTime;
  content = "Time for training + evaluation: " + String.valueOf
     (totalTime) + "\n";
  System.out.println(content);
  Files.write(file, content.getBytes(), StandardOpenOption.
     APPEND);
  startTime = System.currentTimeMillis();
 }

 public static Classifier buildJ48(String options, Path file)
    throws Exception {
```

```
 String[] options_J48 = weka.core.Utils.splitOptions(options);
 content = "==========\n J48 "
   + Arrays.asList(options_J48).stream().collect(Collectors.
       joining(", "));
 Files.write(file, (content + "\n").getBytes(),
     StandardOpenOption.APPEND);

 J48 tree = new J48();
 tree.setOptions(options_J48);
 return tree;
}

public static Classifier buildNaiveBayes(String options, Path
    file) throws Exception {
 String[] options_NB = weka.core.Utils.splitOptions(options);
 content = "==========\n Naive Bayes"
   + Arrays.asList(options_NB).stream().collect(Collectors.
       joining(", "));
 Files.write(file, (content + "\n").getBytes(),
     StandardOpenOption.APPEND);

 NaiveBayes nb = new NaiveBayes();
 nb.setOptions(options_NB);
 return nb;
}

public static Classifier buildKNN(String options, Path file,
    int k) throws Exception {
 IBk ibk = new IBk();
 ibk.setKNN(k);
 String[] options_knn;
 if (options == "")
  options_knn = ibk.getOptions();
 else
  options_knn = weka.core.Utils.splitOptions(options);
 content = "==========\n K-Nearest Neighbour "
   + Arrays.asList(options_knn).stream().collect(Collectors.
       joining(", "));
 Files.write(file, (content + "\n").getBytes(),
     StandardOpenOption.APPEND);

 return ibk;
}
```

```
 public static Classifier buildSVN(String options, Path file)
    throws Exception {
  SMO svm = new SMO();
  svm.setFilterType(new SelectedTag(SMO.FILTER_NONE, SMO.
      TAGS_FILTER));

  String[] options_svn;
  if (options == "")
   options_svn = svm.getOptions();
  else
   options_svn = weka.core.Utils.splitOptions(options);

  content = "==========\n Support Vector
      Machine "
    + Arrays.asList(options_svn).stream().collect(Collectors.
        joining(", "));
  Files.write(file, (content + "\n").getBytes(),
      StandardOpenOption.APPEND);
  return svm;
 }
}
```

Bibliography

Y. Aphinyanaphongs, L. D. Fu, Z. Li, E. R. Peskin, E. Efstathiadis, C. F. Aliferis, and A. Statnikov. A comprehensive empirical comparison of modern supervised classification and feature selection methods for text categorization. *Journal of the Association for Information Science and Technology*, 65(10):1964–1987, 2014. doi: 10.1002/asi.23110.

A. L. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1-2):245–271, 1997. ISSN 00043702. doi: 10.1016/S0004-3702(97)00063-5.

K. J. Cios, W. Pedrycz, R. W. Swiniarski, and L. A. Kurgan. *Data Mining: A Knowledge Discovery Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

S. Drazin. Decision Tree Analysis using Weka. *Machine Learning-Project II, University of Miami*, pages 1–3, 2010.

S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the 7th international Conference on Information and Knowledge Management (CIKM '98)*, pages 148–155, 1998.

G. Forman. An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research*, 3:1289–1305, 2003. doi: 10.1162/153244303322753670.

Q. Guo, W. Wu, D. Massart, C. Boucon, and S. de Jong. Feature selection in principal component analysis of analytical data. *Chemometrics and Intelligent Laboratory Systems*, 61(1-2):123–132, 2002. doi: 10.1016/S0169-7439(01)00203-9.

I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research (JMLR)*, 3(3):1157–1182, 2003. doi: 10.1016/j.aca.2011.07.027.

V. Ha-Thuc and J.-M. Renders. Large-scale hierarchical text classification without labelled data. *Proceedings of the fourth ACM international conference on Web search and data mining - WSDM '11*, page 685, 2011. doi: 10.1145/1935826.1935919.

M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

B. Harish, R. M. Hegde, N. Neeti, and M. Meghana. An Empirical Study on Various Text Classifiers. *Advanced Materials Research*, pages 587–593, 2012. doi: 10.1109/MSR.2017.60.

R. C. Holte. Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, 11(1):63–91, 1993. doi: 10.1023/A:1022631118932.

C. L. Huang and C. J. Wang. A GA-based feature selection and parameters optimization for support vector machines. *Expert Systems with Applications*, 31(2):231–240, 2006. doi: 10.1016/j.eswa.2005.09.024.

Informatica LLC. Portio Research: Worldwide SMS Markets 2014-2017, 2017. URL `https://now.informatica.com/en_daas-portio-research_white-paper_2795.html?asset-id=e74d6383a272c83765fb82f2d2981bdf#fbid=xHbEmWlN2bq`.

A. Janecek, W. Gansterer, M. Demel, and G. Ecker. On the relationship between feature selection and classification accuracy. In *New Challenges for Feature Selection in Data Mining and Knowledge Discovery*, pages 90–105, 2008.

T. Joachims. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. *Proceedings of the 10th European Conference on Machine Learning ECML '98*, pages 137–142, 1998. doi: 10.1007/BFb0026683.

I. T. Jolliffe. Discarding variables in a principal component analysis. I: artificial data. *Applied Statistics*, 21(2):160–173, 1972.

R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997. doi: 10.1016/S0004-3702(97)00043-X.

P. Langley and W. Iba. Average-case analysis of a nearest neighbor algorithm. *IJCAI'93: Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 13:889–889, 1993.

J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, 5:361–397, 2004. doi: 10.1145/122860.122861.

I. Mercur IT Solutions. The Dark Crawler — Exploring the dark corners of the Internet. \url{http://thedarkcrawler.com}, 2017.

J. Michel, Y. Shen, and A. Aiden. Quantitative analysis of culture using millions of Digitized Books. *Science (New York, N.y.)*, 331(6014):176–182, 2011. doi: :10.1126/science.1199644.

M. Oakes. Text Categorization: Automatic Discrimination between US and UK English Using the Chi-square Text and High Ratio Pairs. *Research in Language*, 1:143–156, 2003.

D. L. Olson and D. Delen. *Advanced Data Mining Techniques*. Springer Publishing Company, Incorporated, 1st edition, 2008.

F. Peng and D. Schuurmans. Combining Naive Bayes and n-Gram Language Models for Text Classification. *Computer*, pages 335–350, 2003. doi: 10.1007/3-540-36618-0_24.

J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, Mar. 2002. doi: 10.1145/505282.505283.

F. Sebastiani. Text categorization. In *Encyclopedia of Database Technologies and Applications*, pages 683–687. IGI Global, 2005.

S. Suthaharan. *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*. Springer Publishing Company, Incorporated, 1st edition, 2015. ISBN 148997640X, 9781489976406.

G. R. Weir. Corpus profiling with the posit tools. In *Proceedings of the 5th Corpus Linguistics Conference. University of Liverpool*, 2009.

G. R. Weir and N. K. Anagnostou. Exploring newspapers: a case study in corpus analysis. *Proceedings of ICTATLL 2007*, pages 12–19, 2007.

G. R. S. Weir. The Posit Text Profiling Toolset. *International Journal of Hospitality Management*, 2007.

G. R. S. Weir, E. Dos Santos, B. Cartwright, and R. Frank. Positing the problem: Enhancing classification of extremist web content through textual analysis. *2016 IEEE International Conference on Cybercrime and Computer Forensic, ICCCF 2016*, pages 67–69, 2016. doi: 10.1109/ICCCF.2016.7740431.

I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2011.

E. Witten, Ian H and Frank and Hall. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. *SIGMOD Record*, 31(1), 2002. doi: 0120884070,9780120884070.

X. Zhu. *Knowledge Discovery and Data Mining: Challenges and Realities: Challenges and Realities*. Igi Global, 2007.