Bachelor's Thesis in Artificial Intelligence

Radboud University Nijmegen

Leveraging Text Classification Techniques to Find Events on the Web

Haye Böhm
4290402

supervisors
Dr. F. Grootjen (Radboud University Nijmegen)
Steven Trooster (Ugenda)

reviewer
Dr. L. Vuurpijl (Radboud University Nijmegen)

July 5, 2016

Contents

Abstract

Ugenda is an organization which aggregates event information from other websites. A lot of work goes into selecting web pages which contain events, as there is no fixed structure between different websites. Ugenda is searching for ways of automating the process of event page selection. One approach is to look at the text content of all webpages and automatically determine if the pages contain an event based on the text content.

The text content of web pages is extracted by collecting the text inside select HTML tags. The resulting text is represented by counting the different words in the text and placing those counts in a vector. A dataset is created by crawling (following links on web pages) a select number of websites and performing manual classification (into classes *event* and *other*) in the resulting pages. These pages are then transformed into a format which can be read by the Weka datamining toolkit. Classification is performed by using three different classifiers to achieve the best performance possible. Three different *weighting schemes* are also used in order to enhance performance.

The results are in line with established literature: Classifiers can distinguish reasonable well between pages with events and other pages. However, the performance is not yet good enough for use by Ugenda.

Additionally, a similar case was investigated by assimilating a random sample of non-event web pages (not restricted by the selected websites by Ugenda) and a number of event web pages from multiple websites, where each website only provides a single event page. Pre-processing was done analogously to the previously mentioned process. Classification of this dataset is, on average, more difficult and thus yields worse performance.

Possible improvements are discussed. The document representation could be changed to include phrases or concepts. The classification algorithms can possibly be tuned further and the collected datasets are too small to draw solid conclusions.

Chapter 1

Introduction

Ugenda is one of the leading cultural calendar websites in the area of Nijmegen[1]. Ugenda aggregates content from venues in and around Nijmegen. The organization provides users with a single website where they can get information on cultural events. These events can range from theatre shows and concerts to musicals or local art exhibitions. Ugenda's website also provides reviews of events and interviews with local art enthusiasts in addition to the online calendar containing all local events.

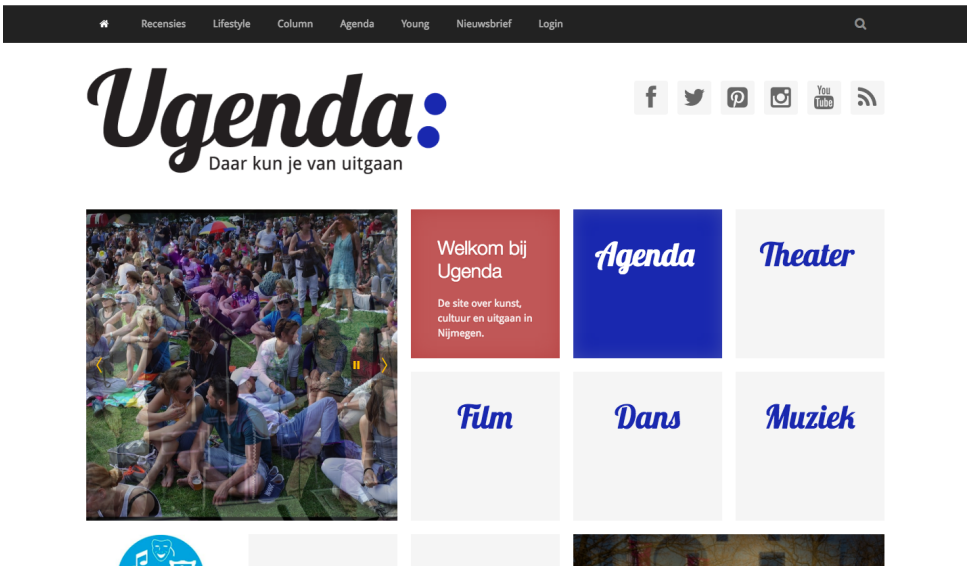

Figure 1.1: A screenshot from Ugenda's website

One of main activities performed by Ugenda is collecting event related information from other websites. Venues put event information on their websites so that their potential customers can keep up-to-date with activities at the venue. Editors from Ugenda have to retrieve this information: they visit the venue websites and manually extract the date, location and description of local events, which they then put in Ugenda's online calendar. There are over 100 different venues, so this task is very time consuming. Although Ugenda has taken some steps to automate this process by using a manually trained web scraper, a lot of progress can still be made to further automate the event extraction process.

[1] `http://www.ugenda.nl`

Ugenda has to go through a number of steps in order to fill their website with events. First, editors identify websites which belong to venues in the Nijmegen area. Every website which has been identified as a venue website then needs to be searched for events. Some websites have a fixed structure, such that events always appear in exactly the same location on the website. Other sites do not have such a structure and extracting events from those websites can be more time-consuming. Lastly, the editors take the pages which contain events and transform the information on the web page into something Ugenda can place on the website. For instance, Ugenda requires a date and time, a description and a category or genre (e.g. musical, concert, theatre show).

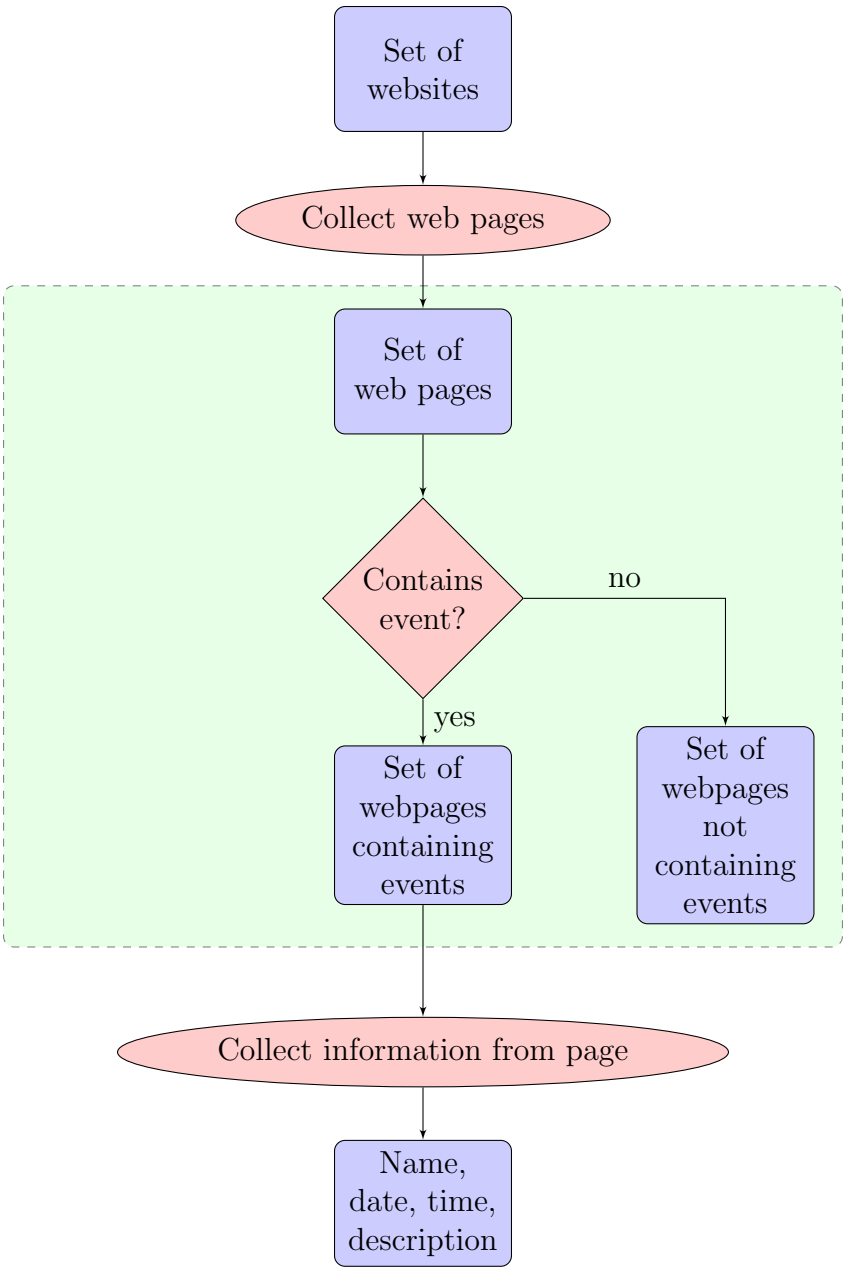

Figure 1.2: A schematic overview of Ugenda's process to collect event information. This is simplified: In reality, editors can use knowledge about the structure of websites to shorten the process.

The process on identifying webpages which contain an event from the set of all web

pages in a domain can be represented as a classification problem [4]. Assuming there is a set of all web pages in and around Nijmegen (similar to those identified by the editors), every web page has to be labeled. In this case, those labels would be "contains event" and "does not contain an event", respectively. See the highlighted part of figure 1.2 for a graphical overview. The process of labeling these web pages is commonly referred to as classification. The aim of this project is to create a program which can automatically classify a set of web pages into a set containing events and a set which does not contain events. Automatic classification has the advantage that no humans are required to find new web pages which contain events, so that Ugenda can focus more of their resources on reviews, interviews, or other content.

There are numerous different approaches to automate this classification problem. For instance, one approach could be to keep track of the HTML structure of the website. The place of relevant information in the tree could be marked, allowing for retrieval across multiple web pages assuming they have the same structure. However, websites do not always have a rigid structure and this structure could be subject to change. Another approach could be to use information available in the Unique Resource Identifier (URL) of the page. For instance, if a URL contains the word "event", it is very likely that the page contains an event.

In this thesis, I will show how I have approached creating a representative data set, converting web pages to a suitable representation and training a classifier to recognize web pages which contain events. I will distinguish two separate cases:

1. Identifying events from a data set containing all web pages from a fixed set of websites

2. Identifying events from a data set containing one event web page for each website, and random other pages.

The first case is the one which is most important to Ugenda. Ugenda already has a list of websites from where they regularly get events. Assuming that all web pages belonging to a website can be retrieved, the classifier needs to identify those pages that contain an event. Ugenda does not have huge computational resources to their disposal, so a proposed solution should not only be effective but also efficient. The solution is regarded as effective if it reduces the amount of web pages the editors have to go through, ideally to only web pages which contain events. The research question for this case is formulated as follows:

> Q_1: *How can a classifier be trained to distinguish between web pages with event and other web pages when the web pages come from a set of domains?*

The second case is the one which is particularly interesting from a research perspective. Can a classifier represent the abstract concepts which belong to an event? This differs from the first case, where the classifier can possibly learn from the structure of event web pages[2], not the content. Thus, the second research question is is formulated as follows:

> Q_2: *How can a classifier be trained to distinguish between web pages with event and other web pages when these web pages are sourced randomly from the internet?*

[2]This can be caused by the fact that a single website has a number of event web pages which are similar in structure. The classifier could then learn that structure, instead of the content of the event (date, time, location)

In chapter 2, I will first provide relevant background literature behind web page classification and text classification along with recent approaches to the both the classification problems. Chapter 3 shows the methods used in building an event classifier. This chapter also shows the process in creating a suitable data set as well as attempts to improve performance. Chapter 4 contains the results for both the research questions mentioned earlier and chapter 5 contains the conclusions to the research questions and discussion.

Chapter 2

Previous Research in Web Page Classification

Web pages are different from text documents. Web pages are based on HTML, a structured language, which is used to define the location and type of different elements on the page. Text documents typically do not have such a structure. Web pages can also contain images, interactive elements and forms. The extra information available on a web page causes classification methods on web pages to differ from classification methods on text documents. However, because I primarily focus on the text content of websites, text classification methods are still relevant when classifying web pages.

Qi [23] identifies three major differences between web page classification and text classification. Text classification research is commonly based on a pre-defined corpus of text, with a fixed authoring style. For instance, title comes first, followed by the introduction, followed by the main bod and a conclusion. In contrast, web pages do not adhere to any style rules and vary wildly in structure (aside from the structure of HTML). Secondly, web pages contain a strict structure which allows the page to be rendered to the screen, whereas any formatting is usually stripped away when a text is inserted into a corpus. Lastly, web pages also contain links to other sites and are placed in a context of other web pages. Text documents typically can not refer to other text documents in the way web pages do.

In order to be able to understand approaches to text or web page classification, it is important to provide a proper formalization of the supervised classification process. Supervised classification entails that the class label for each point in the data set is known before classification, while this is not the case in unsupervised classification.

First, let there be a data set D of size m, which consists of data points in a representation consisting of n features. Let there also be a set of classes C. The class (or multiple classes) of each data point in D is known.

The task of classification now, is to assign some class c from C to each data point in D in such a way that the through classification assigned classes correspond to the known classes as much as possible. The goal is to find a function f which maps from the n input features of each data point to a class from C. Note that it is possible to assign multiple classes to each data point[1]. The idea is that if we can maximize the effectiveness of f (in terms of some evaluation metric, see further down), f can be used to assign a class to a new data record with high probability that the classification is correct.

There are multiple types of classification. Most of the methods mentioned below

[1]Sebastiani [27] refers to this as a "multi-label classification"

perform *hard classification*. A hard classifier assigns one or more classes to each of the input data points. A *soft* classifier assigns a degree of evidence that a input data point belongs to a specific class. Each input document is assigned a set of probabilities, where each probability denotes the chance that the document belongs to a certain class.

2.1 Text Classification

Sebastiani [27] provides an overview of approaches (researched before 2002) in text classification. He distinguishes two different approaches of text classification: A Knowledge Engineering (KE) approach and a machine learning (ML) approach. The knowledge engineering approach consists of manually defining rules on which to classify a document (see equations 2.1 and 2.2 for two examples of rules). This requires that there is an expert who can manually define all rules required in order to achieve a satisfactory performance. The other, more modern approach learns document features from a set of example documents and builds a classifier according to identified features for each class. Because the practical implementation of the machine learning approach is much more feasible in real world applications, this approach has grown in popularity. Sebastiani compares multiple machine learning based approaches, including (but not limited to) Support Vector Machines (SVM), decision trees or rules, and neural networks. The author shows that ML based models can perform similarly to trained professionals, perform well with a large number of documents and can aid existing human classifiers.

$$\text{contains(T, tennis)} \wedge \text{contains(T, football)} \rightarrow is\_sport\_article(T) \tag{2.1}$$

$$\text{contains(T, Mickey)} \wedge \text{contains(T, Mouse)} \wedge \text{contains(T, Minni)} \rightarrow is\_Disney\_article(T) \tag{2.2}$$

2.1.1 Document representation

Sebastiani [27] mentions that the most common approach to model documents in text classification is the use of the "Bag of Words" model. In this model, documents are represented as a vector of the total number of word occurrences in the document. See figure 2.1 for an example. In order to create this Bag of Words every input document needs to be separated into tokens. This process strips away all punctuation and formatting, so that only words remain.

a aa b a b aa	$V_D = (\#a, \#aa, \#b)$
aa b a b b	$V_D = (3, 3, 5)$

Figure 2.1: An example of the Bag of Words model, with the document D on the left and the vector representation of D on the right

More recently, Ikonomakis *et al.* [11] have shown different problems can arise when classifying text documents. First, they shown that classifying an arbitrary number of documents leads to a huge number of different words in the data set. The Bag of Words model mentioned earlier can lead to extremely long vectors, increasing with the size of the input documents *and* the number of input documents. Reducing the length of the document vectors is called Dimensionality Reduction. One of the ways to reduce the dimensionality of text document vectors is stemming (Lovins, 1968 [21], see Willett, 2006

[34] for a more recent approach). Lovins defines stemming as "a computational procedure which reduces all words with the same root". Stemming can reduce the number of distinct words in a document by finding all words which have the same stem. For example, "introduced" and "introducing" have the same stem "introduce".

Another approach is based on the fact that a text document often contains words which do not have direct influence on the content or sentiment of the text. Those words can be conjunction words (and, but, because, therefore, etc. etc.), words which have a high frequency in documents independent of the class or category of the document, or words used frequently in language in general. Removing those words is called stopping. Most recent text classification research involves some form of stopping.

After stemming and stopping procedures, an important step is transforming the input document vectors into term vectors which contain more useful information. A function which transform an input document vector into an more informative vector is called a weighting scheme. A popular weighting scheme is "term-frequency inverse-document-frequency" (*tf-idf*) (*idf* first introduced by Sparck-Jones [29], later extended with *tf* by Salton and McGill [25]).

$$tfidf(t_k, d_j) = \#(t_k, d_j) * log(\frac{|D|}{\#_{Tr}(t_k)}) \tag{2.3}$$

In equation 2.3, $tfidf(t_k, d_j)$ denotes the output of the weighting scheme. t_k denotes the kth term in the jth document d. $|D|$ denotes the total number of documents, whereas $\#_{Tr}(t_k)$ represents the total number of documents which contain term t_k. It is clear from the equation that the *tfidf*-value of a term t in a document d is dependent on the number of times it occurs in d, multiplied by the logarithm of the ratio of total documents and documents containing t. This leads to an intuitive function which decreases when the frequency of a term is large and when that term is also present in a large number of documents. Terms which occur very often in a single document but not in other documents consequently have a high *tf-idf*-frequency, as these terms are likely to be very specific for that document. These high *tf-idf*-frequency terms can be used to describe the document and possibly the class of the document.

Lastly, documents in a data set often have varying lengths. If the previously mentioned weighting scheme were to be used, the weight of a term would be dependent on the length of the document the term occurs in. For example, a term ω which occurs 6 times in a document of length 10 has a *tf* of 6, where as the same term in a document of length 20 would have a *tf* component of 3. Assuming that the *idf* component is constant, the *tf-idf* for ω differs between the two document while they contain ω equally as much. A solution to this is normalization: Add another component into the formula which counteracts the effect of document length differences:

$$tfidf(t_k, d_j) = \frac{\#(t_k, d_j)}{|d_j|} * log(\frac{|D|}{\#_{Tr}(t_k)}) \tag{2.4}$$

where the *tf* component is divided by the length of document j. Note that there are many other forms of normalization with regards to *tf-idf*, for example a different strategy is using the maximum term count, replacing the $|d_j|$ component with the biggest term frequency in document d_j before weighting.

Alternative Document Representation

In addition to the Bag of Words model, different document representation models have been researched. These models aim to add information which is not present in the Bag of Words model, like sequences of words which often occur in a document. However, adding more information can lead to a more sparse document representation which causes the model to be more computationally demanding. The challenge is to manipulate the model in such a way that it is still computationally feasible.

The Bigram document representation model aims to incorporate sequences of words. One of the drawbacks of the Bag of Words model is the fact that it does not take words sequences into account, only word frequency. It effectively ignores information about words groups which occur together frequently in documents. The Bigram model tries to solve this by either adding a number of relevant bigrams calculated from the dataset [30] or using bigrams exclusively. However, extensive research into including phrases (bigrams or phrases of length n n-grams) into the classification process ([33], earlier research by Lewis, 1992 [19]) shows that n-grams only increases performance in few, select cases.

The next step in classifying text documents after a suitable representation is reached is either feature extraction, feature transformation or a combination of both.

2.1.2 Feature Extraction

Feature extraction (also known as feature selection) consists of selecting a number of elements from the input vector in such a way that the resulting vector still contains enough information to successfully classify the vectors. The major advantage of feature extraction is dimensionality reduction, which leads to better classifier performance and reduced over fitting. Feature selection can not be efficiently performed by considering all possible subsets of features. A heuristic should be used to determine the optimal subset of features. Forman [6] shows that the selection of a feature extraction method depends heavily on the input data set and the desired performance (see further down for an overview of analyzing classifier performance).

Forman provides an elaborate overview of feature extraction methods. Relatively simple feature evaluation metrics are described such as *Acc* and *DFreq* as well as methods like Chi-squared or Information Gain. The application of these methods to a data set should lead to a ranking of features on usefulness so that choice can be made to select a number of most useful features. He also proposes a new feature extraction method, Bi-Normal Separation, which uses the statistical z-score in combination with the false-positive ratio and the true-positive ratio of a feature. The false/true-positive ratio of a feature can be defined as the number of documents in the data set which contain this feature (term) and have either a positive classification or a negative classification. This number is then divided by the total number of document with a positive or negative classification.

Forman shows that feature extraction in combination with a SVM classifier only leads to increased performance in a few cases. SVM classifiers already perform well with a large number of input features. The proposed BNS method does provide some performance improvements, however, as it scored better than the other methods on recall, f-measure and accuracy (see the "Performance Evaluation" section).

More recently, Yang *et al.* [35] have shown that a feature extraction method (CMFS) based on the relative occurrence of a term in a category combined with the number of categories and the number of input documents improves performance. Yang shows that

this method often selects features which are also selected by a number of other feature selection method, effectively combining the best features from these methods. Using a Naive Bayes classifier or a SVM, extracting features using CMFS often leads to an improvement in f-measure but not in all cases and not with all data sets.

Chandrashekar and Sahin [1] provide a recent survey on feature extraction methods. Their main conclusion is that using feature selection always benefits the user of classifier. This can be in the form of gaining more information about the data set, improved classifier performance or identification of useless features. They also show that feature selection can be used to enable the use of classification algorithms which usually require lots of computational power, like neural networks.

2.1.3 Feature Transformation

Feature transformation consists of combining elements of the input document vector into a single, new element. A feature transformation function transforms input vectors (data points) into output vectors, which may contain data of the same type as the input data but may also contain data of a new type. An example of this is mentioned by Sebastiani is clustering: transforming a set of input features into a cluster, which aims to represent the concept or similarity between the words.

Another example of feature transformation is Latent Semantic Analysis (or Latent Semantic Indexing). This technique tries to map the terms in a data set into concepts, so that documents can be compared based on the concepts which they contain. This circumvents the problem that words can have multiple meanings, as well as the problem that a concept can be represented by a multitude of words. LSA is performed by observing which words often occur together in a document and then mapping concepts to each of the document in the dataset. Documents can now be compared by the concepts they contain.

However, LSA has two drawbacks. Firstly, LSA requires a large input dataset in order to accurately extract concepts. Secondly, due to the size of the dataset, performing LSA and using it to classify documents takes a large amount of computational resources. A recent interpretation of LSA by Wang *et al.*[32] attempts to solve these two problems by paralellizing the indexing process of concepts and allowing for *stream* calculations of the concepts. This last addition allows for quick processing of documents, saving time and requiring less computational power.

An example of combining feature extraction and transformation can be found in the paper by Uğuz [31]. The goal of the paper is to test the combination of feature selection using Information Gain followed by feature extraction using a Genetic Algorithm followed by Principal Component Analysis. The author shows that combining feature extraction with feature transformation can be very effective.

To recapitulate, after transforming input documents into a suitable representation and optionally performing feature extraction or transformation, we now have a a representation as shown in figure 2.2.

2.1.4 Performance evaluation of Text Classifiers

Before introducing different methods of classification, evaluation of those methods needs to be understood. Text classifiers are evaluated by comparing the output of the classifier on a test set with known classes for the test set. A naive approach to evaluate performance

$$Z_{m,n} = \begin{array}{c|cccc} & t_0 = alpaca & t_1 = bear & \cdots & t_m = zebra \\ d_1 & 0.1 & 0.8 & \cdots & 0.23 \\ d_2 & 0.2 & 0.95 & \cdots & 0.36 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_n & 0.9 & 0.34 & \ldots & 0.12 \end{array}$$

Figure 2.2: Representation of input documents after stemming, stopping, tokenizing and applying a term weighting scheme. Note that m denotes the number of unique terms found throughout all documents and n the number of documents. Terms do not need to be in lexicographical order. Note that this is a fictional example to illustrate the representation.

is to calculate the ratio of correctly classified documents versus the total number of documents. Although this can give a broad idea of how the classifier performs, this accuracy is heavily dependent on the distribution of classes in the test set. If the test set contains 99 documents of class c_1 and 1 document of the class c_2, the classifier would score 99% on accuracy if it simply assigns c_1 to all input documents, without needing to do any classification work. Using accuracy as measure for performance would give the impression that the classifier performs very well even though it does not classify at all. A better approach is to define measurements for every class in the data set, so that performance can be evaluated more precisely.

Two simple measures are most often used: precision (see equation 2.5) and recall (see equation 2.6). These two measures are not independent of the classes in which a classifier needs to classify. Precision for a class c_i can be defined as the number of true positives for that class (the number of documents which are classified under c_1 and also have the label c_1) as a ratio of the sum of the true positives of that class with the false positives for that class. Concisely: The ratio of positive classifications which are correct. A high precision for c_1 implies that the classifier is very good at classifying documents under c_1 correctly.

Recall is similar to precision, with the distinction that it is a ratio of the total number of documents with class c_i which have been classified correctly or have been misclassified. Concisely: The ratio of positive classification which should have been classified as true. A high recall for class c_1 implies that the classifier is able to effectively classify documents into class c_i.

High recall combined with low precision means that the algorithm classifies data points into class c_i effectively, but that the ratio of correct classification for c_i is low. Low recall combined with high precision means that the algorithm misses a lot of document which belong to c_i, but that almost all document classified into c_i actually have the c_i label.

Precision and recall are often combined in the f-measure. This combination allows summarizing the performance of a classifier into a single value. In equation 2.7, precision and recall can be balanced by using a parameter β. This allows for emphasizing either the importance of recall or the importance of precision.

$$precision_{c_i} = \frac{TP_{c_i}}{TP_{c_i} + FP_{c_i}} \tag{2.5}$$

$$recall_{c_i} = \frac{TP_{c_i}}{TP_{c_i} + FN_{c_i}} \tag{2.6}$$

$$f_{c_i} = \frac{(1+\beta^2) * precision * recall}{(\beta^2 * precision) + recall} \tag{2.7}$$

Sokolova and Lapalme [28] provide an elaborate overview of performance measures used in classification. They conclude that performance evaluation metrics differ based on their *invariance* properties, that is if a change in TP/TN/FP/FN numbers leads to a change in the selected evaluation measure. They note that different evaluation metrics should be chosen when classifying human communication (letters, forum posts) as opposed to document classification (newspaper articles, blog posts).

2.1.5 Classification Algorithms

Support Vector Machines
Introduced originally by Cortes and Vapnik [5], Support Vector Machines allow solving complex classification problems. To solve classification problems which are not linearly separable (a single straight line can not be drawn between classes), SVMs use a *kernel trick* in order to map the input vectors into a higher dimensionality space which transforms the problem into a linearly separable one in this higher dimension. By transforming this line back into the original space, a curved line is created which can be used to perform classification.

A function which performs this mapping is called a *kernel*. Multiple kernels are possible, ranging from relatively simple, fast ones like a linear kernel to more computationally expensive kernels like Radial Basis functions. Choosing a kernel appropriate to the classification problem is important, as different kernels achieve different results and require different amount of computational power.

Depending on the chosen kernel, parameters can be adjusted to allow for optimization. For example, the Radial Basis kernel has a γ parameter, whereas a Polynomial kernel has γ, *coefficient* and *degree* parameters. Finding the optimal combination of parameters is often achieved by some form of search: Trying out different combination of values and recording the performance.

Joachim [12] provides a guide for text classification using SVM classifiers. He shows that SVMs perform well in when classifying text documents and can easily accommodate the high dimensionality of the input vectors. He also notes that SVMs tend to find good parameter values automatically when the method from the paper is used.

Regression
Regression is strictly speaking not a classification method. Regression consists of estimating a output real value according to a input vector of real values. In the special case of *logistic* regression the output is not a real value but a set of probabilities. Each member i of the output set of probabilities represents the probability that the input vector belongs to the class i. Text classification can be achieved by assigning the class which corresponds to the highest probability to the input document, often using a threshold.

Logistic regression has proven to be quite effective at text classification. Genkin *et al.* [8] have used logistic regression in combination with a Gaussian distribution to initialize the model parameters. Their Lasso Logistic Regression proved to be effective at categorizing text data, with performance equivalent or better than a SVM in almost all tested data sets. They do seem to hint that their Lasso model takes more time to train

than other models.

Neural Networks
Neural Networks are a popular connectionist [10] model often used to solve complicated learning problems. Their main advantage, the ability to represent abstract concepts, is caused by the presence of hidden layers in the network[2]. A neural network mimics the human brain in terms of architecture, which can provide insights into how the human brain learns and operates.

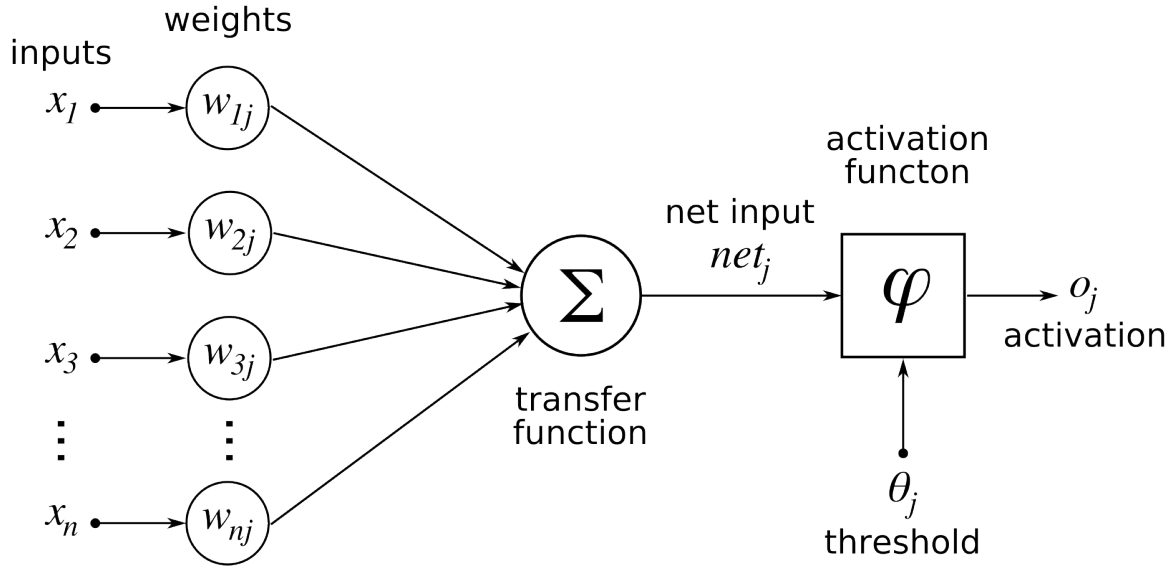

Figure 2.3: A visual representation of a node in a neural network. Originally created by Chrislb, CC BY-SA 3.0, from `https://commons.wikimedia.org/w/index.php?curid=224555`

Neural networks consist of interconnected layers, which have nodes (see figure 2.3). Inputs to the network can, depending on the weight to each input node, cause input nodes in the network to activate if a certain threshold is reached and propagate the activation further into the network. The activation serving as input to a number of nodes in the next layer. Most neural networks have the structure of an input layer, followed by 0 or more hidden layer followed by an output layer.

Nodes are connected with weights to the nodes in the next layer. Updating those weights can change the propagation of activation and thus the output of the network as a whole. Training a neural network is effectively adjusting those weights in such a way that the output of the network matches the expected output as much as possible.

A key disadvantage of a neural network lies in the fact that, without proper hardware, networks are often very slow to train [22] dependent on the size and structure of the network. The size of the input layer of the neural network depends on the size of the input vector. The input vector can get very large in the case of text classification, so proper feature extraction methods or feature transformation methods are required. Lam and Lee [18] have shown however that it is possible to effectively use neural networks in text classification. The authors found the most effective feature transformation method to be Principal Component Analysis. A different disadvantage is that a neural network is

[2]Hidden layers are not a strict requirement of neural networks. They do enable to network to solve linearly inseparable problems

not as easy to interpret by humans as *e.g.* a decision tree. The meaning of the input and output layers can be determined, however the hidden layers do not have a fixed meaning. This meaning is learned from the dataset and is more difficult to understand.

Bayesian Classifiers
Probabilistic learning methods are based on the idea that, given a class c, the values of features can be predicted from the data set. Classification then happens by applying Bayes' rule

$$P(Y = c|F) = \frac{P(F|Y = c) * P(Y = c)}{P(F)} \tag{2.8}$$

where $P(Y = c|F)$ denotes the probability that class c belongs to feature-value set F. A class value is then assigned by selecting the class with the highest probability. By the conditional independence assumption, the denominator can by calculated by summing over $P(w|c)$, for all items w in the feature set.

Major advantages of Bayesian classifiers are speed, ease of implementation and relatively good performance. A disadvantage is the independence assumption. Words in a text are often not independent. In practice, classifiers that do not check this assumption are called Naive Bayes classifiers.

An example of text classification by Chen [2] shows that feature selection can be very effective when using a Naive Bayes classifier.
Decision Trees
Decision trees provide an intuitive, easy to understand way to classify documents into one or more categories. Decision trees work by choosing a term in the document vector and splitting the data into two parts: One part which contains the same value for the chosen term and one with different values. For each of the two splits, the procedure is repeated recursively until all remaining parts only contain documents of the same class.

Decision trees have the advantage that the output can be easily interpreted by humans. Decision trees are also able to deal with varying formats of input data, are relatively quick and often require little data preprocessing. A disadvantage is that decision trees require pruning else they are very sensitive to over fitting. A pruning parameter needs to be determined which takes away from the speed of the algorithm.

Johnson *et al.* [14] use decision trees in conjunction with a new transformation method to decision rules to classify texts. The authors showed that their method could reach a precision of 92.8 percent and a recall 89.1 percent on a single dataset. Their method was also robust and could even classify mislabelled input data into the correct category.

2.1.6 Recent Text Classification Research

Text classification has a wide range of possible applications. For example, Sarker *et al.* [26] have shown that text classification combined with Natural Language Processing can be used to detect symptoms of an adverse drug reaction. They use a SVM classifier and data from Twitter, DailyStrength and an annotated corpus of text to speed up detection of unpredicted reactions to new drugs as they are brought on the market. They also explicitly note that the use of multiple data sources (corpus) can improve performance when the sources contain similar data.

Lai *et al.* [17] use multiple Neural Network types to classify texts. They use a recurrent neural network to be able to store text data inside the network. In order to

account for the fact that terms which occur later in a document have more influence on a recurrent neural network, the authors add a convolutional component which allows fair classification without putting more emphasis on words which occur later. They report that their model performs better than classic text classification methods (*e.g.* a Bag of Words approach combined with logistic regression), while also being quick to train on a relatively basic computer.

2.2 Web page classification

Web page classification is different from text classification, because of the availability of more information and more diverse information. Classifiers can potentially use the URL of the web page, the frequency of HTML tags, the content of the tags, etc. etc. The multitude of potential inputs for a classifier has lead to a big number of approaches.

Kan [15] uses features extracted from the URL of a web page. He shows that, using Maximum Entropy classification or a SVM, URL features can be used to successfully classify web pages into 4 categories. He achieves similar performance to text-based classification with an accuracy of about 76% using only information extracted from the URL and and accuracy of about 81% when adding the text content of the web page.

A web page classification approach closely related to text classification is the approach by Riboni [24]. Just like in text classification, a document is represented by the words contained in the document. However, due to the fact that HTML is structured and the assumption that some elements are more important than others, the terms can be weighted according to the location of the term. This Structured Weighting Approach assigns a bigger weight for a document term when in occurs in a HTML tag considered to be more important like META, TITLE and various headings. The author shows that this weighting technique improves performance when compared to classical text based weighting techniques.

Chen and Hsieh [3] use combine two separate classifiers to classify a text representation of websites. One classifier uses latent semantic analysis to provide a suggested class, where the other uses a web page feature selection method to provide a classification. These two classifications are then combined into a final judgement. The authors show that their voted classification approach outperforms other classification methods, even in a relatively small data set.

Chapter 3

Methods

I have chosen to use the Weka datamining toolkit [9] to perform classification. Weka provides a wide range of classification algorithms and data manipulation tools and filters. This allows for easy comparison and experimentation.

3.1 Dataset

First, a suitable dataset needs to be created. It should be a representative sample of web pages encountered during web crawling. It should also be fairly large: Performance of a classifier can only be measured realistically when there is no under or over fitting in the dataset or when measures have been takes to prevent under or over fitting.

3.1.1 RQ1: identifying events in a domain

Given that Ugenda is interested in events in the area of Nijmegen, this dataset only has to contain web pages from around Nijmegen. Ugenda has provided a list of domains in which they are interested. I have used a Bash-script to crawl these domains and extract all web pages from the web server. Due to memory constraints, the script only visits pages which are at maximum three links from the main website. The script only visits and saves those links which are on the same domain as the input website, in order to prevent links to scripts, advertisements and other websites. See code listing 3.1 for the script.

The script outputs a file with a sorted list of URLs encountered in the domain. After some examination, some of these URLs can quickly be classified as not containing an event due to the way the URL is formatted (See figure 3.1 for some examples). For all other websites, the URLs need to be manually visited and filtered into events and other web pages.

After manual classification, text was extracted using a Java program. For every link, the program retrieves the HTML, finds all HTML tags which usually contain text (`<p>`, `<h>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`) and saves the text content to a file. The program maintains the separation between events and other web pages.

As the dataset needs to be used with the data mining toolkit Weka, all separate files need to be converted to a format that Weka can use directly. Weka provides a utility program which converts a directory of text into the format accepted by Weka[1] called

[1] Attribute-Relation File Format (ARFF), see `http://weka.wikispaces.com/ARFF` for an overview

```
#!/usr/bin/env bash
start_url=$1
domain=$(echo $1 | awk -F/ '{print $3}')
touch "output/$domain"
echo "outputfile: output/$domain"

output="$(wget -e robots=off --force-html --spider --
   recursive --level=3 --convert-links -H --domains=$domain
   --page-requisites --html-extension $start_url 2>&1)"
echo "Done crawling.."
#grep "^\-\-" | awk '{ print $3 }'
echo "${output}" | grep "^\-\-" | awk '{ print $3 }' | grep -
   v "\.gif\|png\|jpg\|css\|\.js\|JPG\|ico\|\.pdf$" | sort |
   uniq > "output/$domain"
```

Listing 3.1: The script used to crawl websites. It can be used by invoking the script with full web address as the first argument: "./crawler.sh http://www.lux-nijmegen.nl/"

- `http://www.delindenberg.com/winkelwagen/toevoegen/44028?ie9=3398690934`
- `http://www.lux-nijmegen.nl/info/over-lux/`
- `http://www.museumhetvalkhof.nl/bezoekinformatie.html`

Figure 3.1: Some urls which can not be classified as an event solely from the URL

TextDirectoryLoader.

With the data in ARFF format, the text data can be converted to a document vector using the built in filter StringToWordVector. This filter has a lot of options which can be used to simplify text classification. I configured the filter to remove stopwords from the dataset[2], to output word frequencies and to differentiate between capital and normal letters. The filter can also "try to keep n words", which I set to 5000 due to memory limitations (The program has to be able to fit the whole set into memory). Applying a filter lead to a data set like mentioned in figure 2.2: A vector of words and their frequencies with a length of about 5000.

I order to evaluate if the classifier is overfitting, I also included some random pages from Wikipedia. If the performance were to dramatically decrease after adding the pages, it is probable that the algorithm overfits[3]. I have chosen to create 5 different datasets, each with a different set of Wikipedia articles ranging from 0 articles (the original dataset) to 1100 added articles. The articles were retrieved by using the *Special:Random*[4] function that Wikipedia provides. Note that the articles retrieved were in Dutch.

[2]See appendix 5.1.4 for the words used

[3]This is not really a problem for *RQ1*, as the set of pages contains a fixed number of websites

[4]By visiting `https://en.wikipedia.org/wiki/Special:Random`, Wikipedia redirects to a random article

Dataset	Events	Other	Wikipedia Articles	Number of Attributes (terms)
1	1203	5445	0	5105
2	1203	5445	300	5054
3	1203	5445	500	5129
4	1203	5445	800	5109
5	1203	5445	1100	5058

Table 3.1: Overview of the datasets used in case 1. The number of attributes differ between the datasets due to the way the StringToWordVector filter works.

3.1.2 RQ2: identifying abstract event page properties

The creation of the data set for the second case two in very similar to case one, with a few important differences:

- Events were crawled "by hand": I manually searched for Dutch venues and theaters and saved the event URLs to a file. I only selected one distinct event page for each venue.
- The other web pages were obtained through the use of Google and a Dutch dictionary. I selected a number of random words from the dictionary. For each of those words, I used the Google search engine with the locale and location set to Dutch to retrieve about 10 web pages. This is repeated until the desired number of web pages has been achieved. This process was automated.

After obtaining the URLs, the rest of the procedure is exactly the same as that for case 1. The dataset consisted of 187 different event websites and 5057 random websites from Google.

3.2 Classifier selection

Literature shows that different datasets yield different results depending on classifiers, that is there is no single classifier which performs best on all datasets. This is why I selected three classifiers: a Naive Bayes classifier, a SVM (of C-SVC type) and a C4.5 Decision Tree classifier. By performing classification with three classifiers I hope to find the one that performs best.

I chose these specific classification methods because literature shows they can be trained quickly, perform well and in the case of SVM and Decision Tree can be tuned to enhance performance.

3.3 Training, Optimizing and Evaluation

I trained the classifiers by feeding the different datasets into Weka and performing classification with one of the three selected classifiers. In the case of SVM, I performed GridSearch to search for the best combination of *cost* and ε values. This yielded the best performance at a cost value of 1.0 and an ε value of $1 * 10^{-4}$. The training and evaluation was performed on a computer with a 3.4 GHz Core i5 4690 CPU with about 6 gigabytes of RAM available for the Weka toolkit.

A baseline performance was obtained by using the raw, non-normalized frequencies as input to the classifier. In addition, a *tf-idf* weighting scheme was used to enhance performance as is common in text classification research.

After evaluating the performance of these two weighting schemes, I added a third scheme developed by Liu *et al.* [20]. This scheme aims to enhance performance of unbalanced datasets by including information about class membership. The weighting formula can be expressed as follows:

$$w(i,j) = tf(t_i, d_j) * log(1 + \frac{A}{B} * \frac{A}{C})$$

where:

$A =$ Number of documents belonging class c_i where t_i occurs at least once

$B =$ Number of documents not belonging to class c_i where t_i occurs at least once

$C =$ Number of documents of category c_i which do not contain t_i

This scheme enhances the *tf-idf* scheme intuitively: The *idf* component is replaced by a component which is dependent on the distribution of terms across classes. This component increases in value when a term has a high predictive value for a class, that is A is high while B and C are low.

The results of the classification algorithm are be analysed by comparing the f-measure, recall, and precision value for each classifier for each dataset. In order to get representative results, I used 10-fold cross validation which negates the effects of an overfitted classifier on performance evaluations. n-folds cross validation works by dividing the dataset into n different subsets. The classifier is then trained using $n-1$ sets as training set and one of the remaining set as test set. This is repeated n times, so that each set has been used as test set exactly once.

Observing that the dataset is quite unbalanced (the ratio of events to other pages is small) cost weighted evaluation and sampling can be used to improve performance. However, cost-weighted evaluation (the notion that each type of classification error made by the classifier has a different weight and that the classifier can be optimized according to the sum of these weights) did not yield significant improvements. Sampling (selecting or re-using documents from the dataset) also did not improve performance.

In addition to the methods described above, I have tried a number of different optimization techniques to see if they would improve performance. Feature extraction using either Gain ratio as subset evaluator, Principal Component Analysis or Latent Semantic Analysis did not yield promising results. PCA and LSA both took extremely long, in the case of LSA never terminating without running out of memory. The Gain ratio subset evaluator did result in a optimal subset of parameters though this set decreased classifier performance.

A classifier which I expected to yield good results was a Multilayer Perceptron (a simple neural network). The parameters which can be optimized here are the number of hidden layers and the learning rate. Increasing the number of hidden layers seemed to improve performance marginally. However, increasing the number of hidden layers lead to an increase in training time. I stopped experimenting with a Multilayer Perceptron when I could not approach the performance of a SVM in approximately the same training time, and experimenting became difficult.

Combining multiple classifiers using AdaBoost [7] also proved to be too time consuming.

Chapter 4

Results

4.1 RQ1

The measured *f*, recall and precision scores can be found in table 4.1. All classifiers performed above chance level.

Classifier	Wiki	CWS			tf-idf			Frequencies		
		f-value	p	r	f-value	p	r	f-value	p	r
Bayes	0	0,728	1	0,573	0,644	0,525	0,832	0,618	0,771	0,515
SVM		0,728	1	0,572	0,727	0,801	0,666	0,523	0,364	0,929
C4.5 Tree		0,725	1	0,569	0,704	0.922	0,633	0,737	0,661	0,832
Bayes	300	0,677	1	0,512	0,637	0,517	0,831	0,616	0,756	0,52
SVM		0,734	1	0,58	0,728	0,805	0,665	0,537	0,378	0,929
C4.5 Tree		0,734	1	0,58	0,707	0,797	0,635	0,745	0,672	0,836
Bayes	500	0,666	1	0,499	0,629	0,508	0,827	0,602	0,718	0,519
SVM		0,753	1	0,604	0,733	0,808	0,671	0,543	0,383	0,937
C4.5 Tree		0,752	1	0,603	0,708	0,791	0,641	0,757	0,7	0,825
Bayes	800	0,632	1	0,462	0,615	0,488	0,831	0,602	0,718	0,519
SVM		0,744	1	0,593	0,726	0,805	0,661	0,542	0,386	0,914
C4.5 Tree		0,743	1	0,591	0,708	0,799	0,636	0,736	0,66	0,83
Bayes	1100	0,592	1	0,42	0,581	0,448	0,827	0,595	0,743	0,496
SVM		0,737	1	0,583	0,729	0,806	0,665	0,53	0,37	0,933
C4.5 Tree		0,735	1	0,581	0,703	0,795	0,631	0,725	0,642	0,832

Table 4.1: Raw results from research question one. Note that *Frequencies* denotes the raw frequencies, not processed by any weighting scheme. The *Wiki* column denotes the amount of added Wikipedia articles.

Plotting the results shows that the SVM and Decision Tree classifier are not really affected by the added Wikipedia articles. The naive Bayes classifier shows a small decrease in performance which may be attributes to the addition of Wikipedia articles.

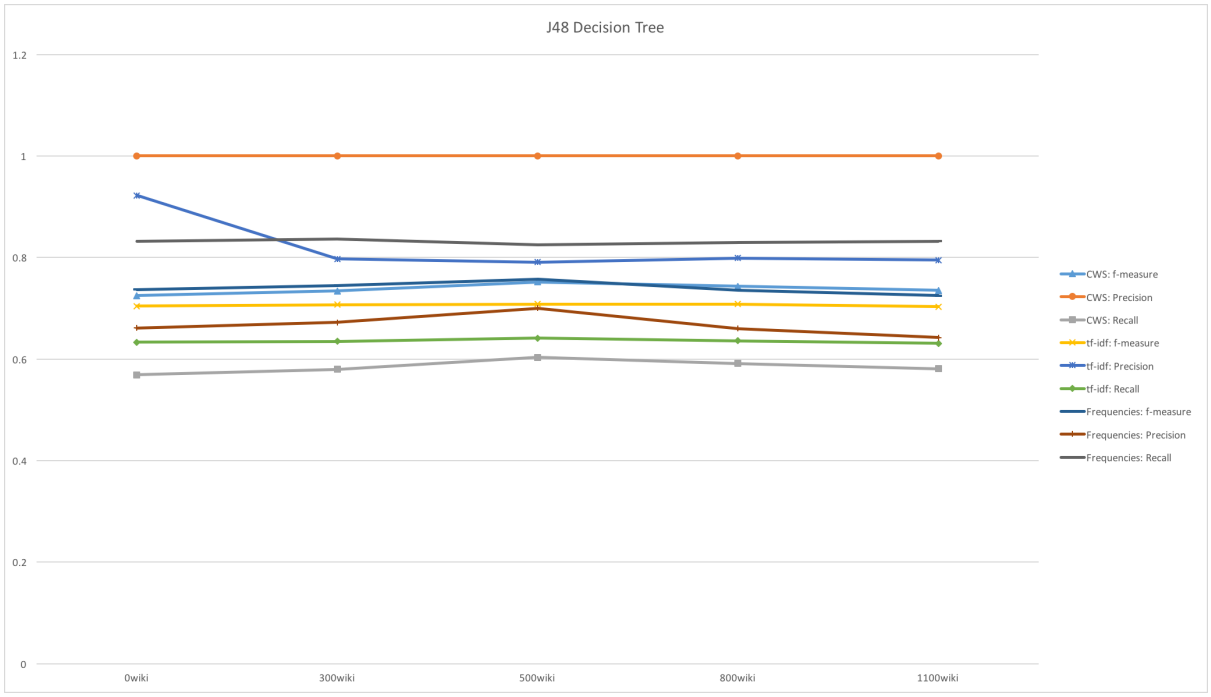

Figure 4.1: Results for the C4.5 decision tree classifier

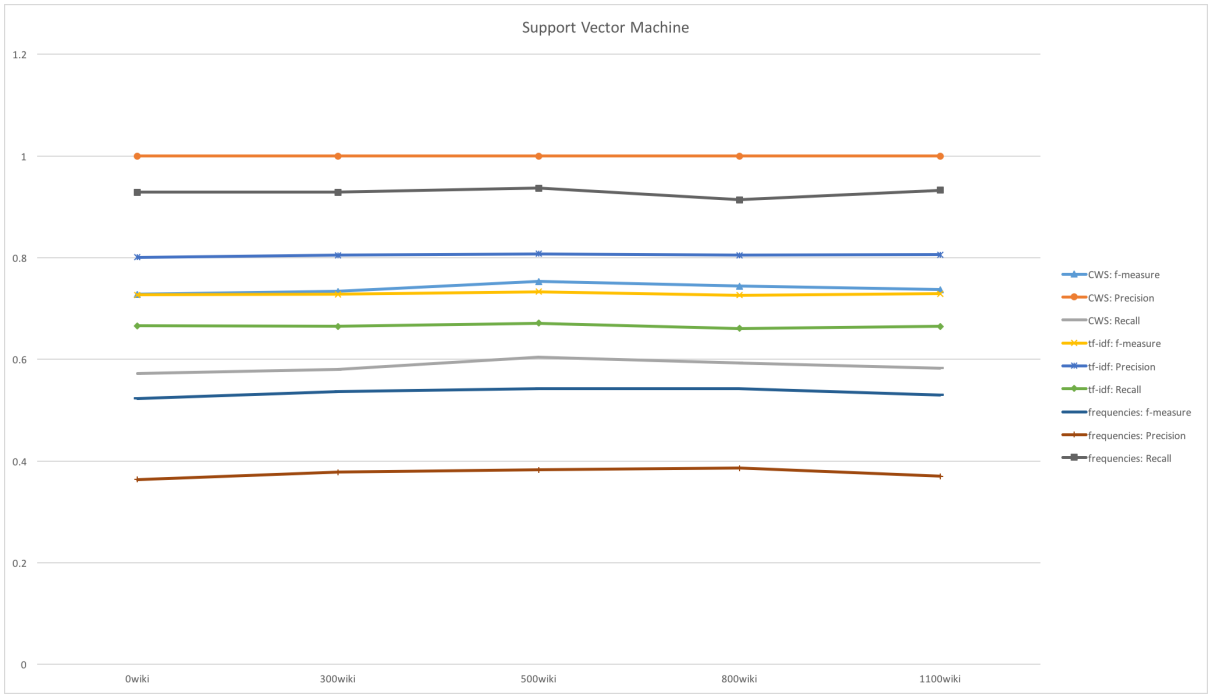

Figure 4.2: Results for the Support Vector Machine classifier

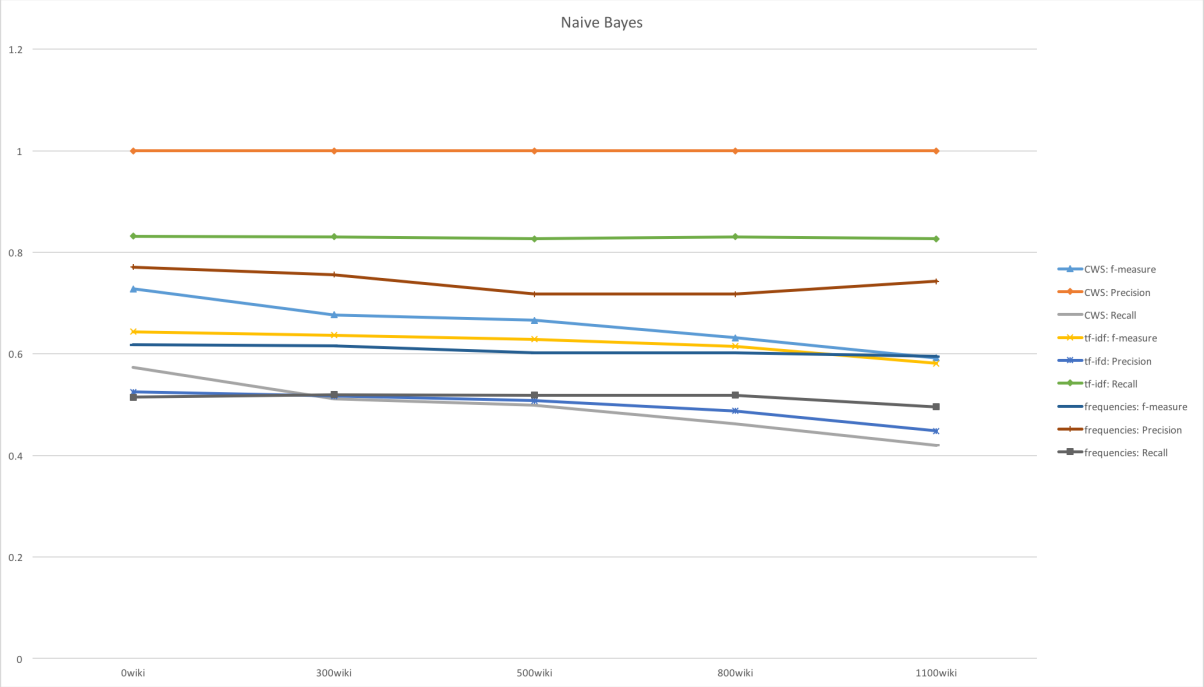

Figure 4.3: Results for the Naive Bayes classifier

	CWS	*tf-idf*	Frequencies
Average f-measure	**0,712**	0,685	0,627
Average Precision	**1,000**	0,708	0,595
Average Recall	0,555	0,710	**0,758**

Table 4.2: Performance averages over the used weighting scheme

Overall, the custom weighting scheme performed best when measuring performance using the f-measure, due to the fact that the precision is always 1,000. However, when using recall as evaluation metric both *tf-idf* and raw frequencies outperform the custom weighting scheme.

	Bayes	SVM	Dtree
Average f-measure	0,6259	0,6676	**0,7385**
Average Precicion	**0,8321**	0,7271	0,8006
Average Recall	0,5064	0,7268	**0,7314**

Table 4.3: Peformance averages over the used classifier

Table 4.3 shows that the C4.5 decision tree and the naive Bayes classifier perform best based on averages. The C4.5 decision tree achieves the best f-measure and recall, the Naive Bayes classifier the best precision.

The different classifiers vary in terms of training time. The Naive Bayes' classifier was the fastest (training took about 5 seconds), followed by the SVM with a training time around 10 seconds. The Decision Tree took longer in the cases of the *frequencies* and *tf-idf* weighting schemes, but was almost as fast in the case of the custom weighting scheme. This was also reflected in the size of the tree, which was decreased by a factor 10 when applying the custom weighting scheme.

4.2 RQ2

	Bayes	SVM	Dtree
Average f-measure	0,107	**0,682**	0,460
Average Precision	0,057	**0,857**	0,667
Average Recall	**1,000**	0,682	0,351

Table 4.4: Result when using the *tf-idf* weighting scheme

	Bayes	SVM	Dtree
Average f-measure	0,128	**0,701**	0.505
Average Precision	0,069	**0,850**	0.632
Average Recall	**0,947**	0,596	0.421

Table 4.5: Result when using no weighting scheme

	Bayes	SVM	Dtree
Average f-measure	**0,163**	0,000	0,000
Average Precision	**0,089**	0,000	0,000
Average Recall	**1,000**	0,000	0,000

Table 4.6: Result when using the custom weighting scheme

The tables show that a SVM classifier performed best in the *tf-idf* case and when no weighting scheme was applied. It is remarkable that applying the custom weighting scheme causes the SVM and decision tree to be unable to classify anything. Also remarkable is the fact that applying the *tf-idf* scheme reduced performance compared to no weighting scheme (from a size of about 200 to 20) .

Chapter 5

Discussion

5.0.1 Conclusion RQ1

The results show that the performance of the classifier is comparable with what can be found in the literature. In the fixed-domain case, Support Vector Machines performed well when considering that recall is more important to Ugenda than precision[1]. A classifier with high recall and mediocre precision can still be used to reduce the input dataset in size.

The custom weighting scheme from [20] has improved precision in all cases except one: the case which combined a C4.5 tree with a *tf-idf* weighting scheme.

Overall, performance between datasets and classifiers varies. There is no single classifier which performs the best across the board. There is also no weighting scheme which performs best across the board. This leads to the conclusion that the choice of classifier and weighting scheme must be made based on the dataset which they are applied on and the results obtained in the previous section.

With regards to Ugenda, it seems that an Decision Tree classifier is most appropriate considering the high average recall, especially combined with the custom weighting scheme. This insures that the amount of web pages Ugenda needs to examine is reduced. However, a lot of improvements need to be made in order for an effective application in Ugenda's workflow.

5.0.2 Conclusion RQ2

A SVM classifier works reasonably well in identifying events. One observation of note is that the CWS did not yield any useful results when using the SVM or C4.5 classifier. The results show that it is possible to distinguish event pages from other web pages. Further research should be done to improve performance.

5.1 Discussion

There are a some discussion points to be considered regarding the dataset, approach and chosen classifiers.

[1] A high recall implies that all events in the dataset are actually classified as events. This ensures that no events are missed.

5.1.1 Dataset

First, both cases could suffer from a lack of data, either in the number of attributes (terms) or the number of documents. Datasets commonly used in the literature, for example the Reuters-21578 dataset which contains 21758 documents, often contain more documents. However, due to the fact that the type of classification in this thesis is binary classification, the number of documents in the dataset may be sufficient. A dataset needs to contain sufficient samples for each class for it to be able to accurately represent that class. There are fewer classes in binary classification, which can result in a lower total number of documents required in order to represent all classes.

The datasets used in *RQ1* could be extended. Currently, they contains 6 different websites which provide about 5000 web pages. Adding more websites to a dataset can lead to a less overfitted classifier, a better idea of how the classifier would perform in reality and possibly better performance. Performance increases could be achieved when, by virtue of the added websites to the dataset, the classifier starts to recognize the features of events, not the features of the structure of web pages containing events.

The Wikipedia articles which were added to this dataset aimed to give an idea about the level of overfitting in the dataset. There was a slight trend in the results which indicated that more Wikipedia articles decreased performance. It should be investigated whether this is also the case when more than 1100 articles are added to the database. Adding other web pages to evaluate the level of overfitting is a possibility, although then the dataset is very similar to the one used in *RQ2*.

A similar point to the one made earlier can be made for the second dataset, where the number of events pages is very small. The number of event pages should be increased, so that the classifier can learn the abstract event concepts like date, time and place. The way the pages are collected can also be improved. The *RQ2* dataset now contains a lot of Dutch dictionary websites due to the fact that the queries are words from a dictionary. This could have the effect that the dataset is not representative for web pages encountered during random web crawling.

5.1.2 Web page extraction

The web page extractor program only extracts a small set of HTML elements from the web page. Elements which do not commonly contain text are not extracted from the web page, while these elements could contain text which is important for the classification process. Including this lost information could improve (or degrade) classification performance. Perhaps some visual selection method, which extracts the text which is shown on screen, could provide a more accurate textual representation of the page.

The script used to collect the web pages follows three links from the base website. For example, the program follows a “More Info” link on the first page, a “Accomodations” link on the second page and a “Appartments” link on the third page. It collects the URLs of pages visited, but does not follow links on the third page. This limitation was imposed to decrease the number of web pages found so that they could still all fit in computer memory. This limitation could cause the removal of important webpages and further research should be done without limitations on the *crawling depth*.

5.1.3 Classifiers

Optimizing the classifiers (where applicable) was done by systematically trying different combinations of parameters and choosing the best combination. It is possible that the found parameters for SVM are not yet optimal. A similar observation should be made with regards to the decision tree classifier. The tree which resulted from the training process should be pruned before it can be applied to real world, new data. This will ensure that the tree is not over fitting. The relatively good performance in case one may be caused by the fact that the tree is not pruned enough.

5.1.4 Possible Improvements

There is a range of possible improvements which could be made to improve performance and power:

- Neural networks are very promising. Used as classifiers they are able to represent abstract concepts and have proven to be effective at text classification [18] when combined with a dimensionality reduction technique. Given more powerful hardware and more time, neural networks could be used to effectively classify web pages and extract events.

- A characteristic of the datasets used in both cases is that they are both unbalanced: The number of events is much lower than the number of non-events. Aside from dataset manipulations (sampling) or evaluation techniques (cost weighted evaluation), the classifier can also be adapted to solve this problem. Kondratovich [16] and Joachims[13] have shown that a different type of SVM classifier can be used to tackle the problem of unbalanced data sets. This *TSVM* classifier can learn from a relatively small sample of data and extend the learned features to a bigger dataset, allowing for better performance when classifying unbalanced data sets.

- Latent Semantic Analysis has been used as a feature transformation method to find abstract concepts found in documents instead of terms and their frequencies. As mentioned in the methods section, LSA proved to be very computationally intensive with the given dataset and was consequently discarded as effective or efficient optimization technique. It would be useful to see whether using LSA would improve performance given enough time and computation power.

- All methods mentioned above assume a Bag of Words document representation model. A different representation, like a bigram or n-gram representation could improve performance. However, literature shows that the expected performance increase is not very high. In addition, finding n-grams and including them in a dataset increases the dimensions of the input vectors, increasing the training time.

Bibliography

[1] Chandrashekar, G., and Sahin, F. A survey on feature selection methods. *Computers & Electrical Engineering 40*, 1 (2014), 16–28.

[2] Chen, J., Huang, H., Tian, S., and Qu, Y. Feature selection for text classification with naïve bayes. *Expert Systems with Applications 36*, 3 (2009), 5432–5435.

[3] Chen, R.-C., and Hsieh, C.-H. Web page classification based on a support vector machine using a weighted vote schema. *Expert Systems with Applications 31*, 2 (2006), 427–435.

[4] Clancey, W. J. *Classification problem solving*. Department of Computer Science, Stanford University, 1984.

[5] Cortes, C., and Vapnik, V. Support-vector networks. *Machine learning 20*, 3 (1995), 273–297.

[6] Forman, G. An experimental study of feature selection metrics for text categorization. *Journal of Machine Learning Research 3*, 1 (2003), 1289–1305.

[7] Freund, Y., and Schapire, R. E. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory* (1995), Springer, pp. 23–37.

[8] Genkin, A., Lewis, D. D., and Madigan, D. Large-scale bayesian logistic regression for text categorization. *Technometrics 49*, 3 (2007), 291–304.

[9] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: an update. *ACM SIGKDD explorations newsletter 11*, 1 (2009), 10–18.

[10] Hinton, G. E. Connectionist learning procedures. *Artificial intelligence 40*, 1 (1989), 185–234.

[11] Ikonomakis, M., Kotsiantis, S., and Tampakas, V. Text classification using machine learning techniques. *WSEAS Transactions on Computers 4*, 8 (2005), 966–974.

[12] Joachims, T. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning* (1998), Springer, pp. 137–142.

[13] Joachims, T. Transductive inference for text classification using support vector machines. In *ICML* (1999), vol. 99, pp. 200–209.

[14] Johnson, D. E., Oles, F. J., Zhang, T., and Goetz, T. A decision-tree-based symbolic rule induction system for text categorization. *IBM Systems Journal 41*, 3 (2002), 428–437.

[15] Kan, M.-Y., and Thi, H. O. N. Fast webpage classification using url features. In *Proceedings of the 14th ACM international conference on Information and knowledge management* (2005), ACM, pp. 325–326.

[16] Kondratovich, E., Baskin, I. I., and Varnek, A. Transductive support vector machines: Promising approach to model small and unbalanced datasets. *Molecular Informatics 32*, 3 (2013), 261–266.

[17] Lai, S., Xu, L., Liu, K., and Zhao, J. Recurrent convolutional neural networks for text classification. In *AAAI* (2015), pp. 2267–2273.

[18] Lam, S. L., and Lee, D. L. Feature reduction for neural network based text categorization. In *Database Systems for Advanced Applications, 1999. Proceedings., 6th International Conference on* (1999), IEEE, pp. 195–202.

[19] Lewis, D. D. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval* (1992), ACM, pp. 37–50.

[20] Liu, Y., Loh, H. T., and Sun, A. Imbalanced text classification: A term weighting approach. *Expert systems with Applications 36*, 1 (2009), 690–701.

[21] Lovins, J. B. *Development of a stemming algorithm.* MIT Information Processing Group, Electronic Systems Laboratory Cambridge, 1968.

[22] Pal, S. K., and Mitra, S. Multilayer perceptron, fuzzy sets, and classification. *Neural Networks, IEEE Transactions on 3*, 5 (1992), 683–697.

[23] Qi, X., and Davison, B. D. Web page classification: Features and algorithms. *ACM Computing Surveys (CSUR) 41*, 2 (2009), 12.

[24] Riboni, D. *Feature selection for web page classification.* na, 2002.

[25] Salton, G., and Buckley, C. Term-weighting approaches in automatic text retrieval. *Information processing & management 24*, 5 (1988), 513–523.

[26] Sarker, A., and Gonzalez, G. Portable automatic text classification for adverse drug reaction detection via multi-corpus training. *Journal of biomedical informatics 53* (2015), 196–207.

[27] Sebastiani, F. Machine learning in automated text categorization. *ACM computing surveys (CSUR) 34*, 1 (2002), 1–47.

[28] Sokolova, M., and Lapalme, G. A systematic analysis of performance measures for classification tasks. *Information Processing & Management 45*, 4 (2009), 427–437.

[29] Sparck Jones, K. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation 28*, 1 (1972), 11–21.

[30] Tan, C.-M., Wang, Y.-F., and Lee, C.-D. The use of bigrams to enhance text categorization. *Information processing & management 38*, 4 (2002), 529–546.

[31] Uğuz, H. A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm. *Knowledge-Based Systems 24*, 7 (2011), 1024–1032.

[32] Wang, Q., Xu, J., Li, H., and Craswell, N. Regularized latent semantic indexing: A new approach to large-scale topic modeling. *ACM Transactions on Information Systems (TOIS) 31*, 1 (2013), 5.

[33] Wang, S., and Manning, C. D. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2* (2012), Association for Computational Linguistics, pp. 90–94.

[34] Willett, P. The porter stemming algorithm: then and now. *Program 40*, 3 (2006), 219–223.

[35] Yang, J., Liu, Y., Zhu, X., Liu, Z., and Zhang, X. A new feature selection based on comprehensive measurement both in inter-category and intra-category for text categorization. *Information Processing & Management 48*, 4 (2012), 741–754.

Appendix A: Stopwords

aan	ed	inzake	ook	uit
afd	een	is	oorspr	uitg
als	en	je	op	vakgr
bij	enige	met	over	van
dat	enkele	na	pas	vanaf
de	enz	naar	pres	vert
den	et	nabij	prof	vol
der	etc	niet	publ	voor
des	haar	no	sl	voortgez
deze	het	nu	st	voortz
die	hierin	of	te	wat
dit	hoe	om	tegen	wie
dl	hun	onder	ten	zijn
door	ik	ons	ter	
dr	in	onze	tot	