Movie reviews: do words add up to a sentiment?

Richard Berendsen

September 14, 2010

Supervisors:
Dr. Marco Wiering, Faculty of Mathematics and Natural Sciences
Dr. Gosse Bouma, Faculty of Arts

rijksuniversiteit
groningen

Abstract

Sentiment analysis, the automatic extraction of opinion from text, has been enjoying some attention in the media during the national elections. In this thesis, we will discuss the classification of movie reviews as 'thumbs up' or 'thumbs down'. Movie reviews are interesting and difficult because of the wide range of topics in movies. The reviews are HTML web pages, which poses an interesting challenge for preprocessing and noise removal. We describe the reviews as 'bags of words' and use support vector machines (SVMs) for classification, as well as transductive support vector machines, which require less training data. To model topics in the reviews, a latent semantic analysis (LSA) was done on a large set of movie reviews. The results show that it is hard to improve SVM performance with latent semantic analysis. The discussion of the results provide some insights into why no performance increase was achieved.

Contents

Chapter 1

Introduction

Suppose you want to know what the general opinion expressed in documents on the Internet is on the PVV, a Dutch political party led by Geert Wilders. You could issue a query to Google containing the words "PVV" and "Geert Wilders". For simplicity, let us assume that the documents returned are indeed on topic. Also, let us assume that there are only positive and negative documents. Now you could download the documents, and classify some documents as positive and negative by hand. Would that allow you to classify the remaining documents automatically and reliably? Furthermore, would it be possible to identify the main topics in the documents? Can knowing these topics help to predict the sentiment of the documents?

In a nutshell, these are the questions we address in this research. Only we do not download documents returned by Google about some controversial political party. Instead, we focus on movie reviews. Movie reviews are interesting and difficult, because of the wide range of topics in movies. Like in real life, like in politics, in movies any subject can play a role. Previous research on movie reviews has been done, and datasets are available. And of course, movie reviews are fun to work with. We use a dataset of 1000 positive and 1000 negative movie reviews that was developed by Pang & Lee (2004). The task that we address is: can we automatically predict if a given review is positive or negative?

Now that we have introduced what it is we do, we will discuss some related work in the next section. There we will see that we want to use support vector machines because of their great performance. And we will hear that latent semantic analysis can be used to model topics and to calculate a semantic similarity. We end this introduction with an informal statement of our research questions. In chapters 2 and 3 we discuss the theory behind support vector machines and latent semantic analysis. This will allow us to state our research questions more precisely as we discuss our experiments. In the last chapter, we draw conclusions and describe possible directions for future work.

1.1 Related work

Automatically predicting if a review is positive or negative is a kind of opinion mining. It can also be called sentiment analysis. Pang & Lee (2008) give a good review of the field. Here we will discuss some related work in predicting movie reviews, and motivate our choices about which technologies we use. If some terms are unfamiliar to you in this section point, you can decide to first read the chapters with theoretical background. There we discuss the ideas mentioned here in an easy to understand way.

1.1.1 Unsupervised approaches

Turney et al. (2002), in an early work, tried to predict the sentiment of car, bank, travel and movie reviews. He uses an unsupervised algorithm that extracts specific two word part of speech patterns from review text and calculates a 'semantic orientation' for them that states whether they are more related to the word 'excellent' or to the word 'poor'. He uses pointwise mutual information (PMI) between the two word patterns and the words excellent and poor, calculating them with the aid of the AltaVista search engine. He calls this approach PMI_IR, the IR for information retrieval. The average semantic orientation of a review is subsequently used to predict whether the review is positive or negative. This approach was tested for several domains and obtained over eighty percent performance for reviews of cars and banks. However, Turney noted that the performance was worst in movie reviews, with 65.83%. What makes classifying movie reviews as positive or negative difficult?

semantic orientation

Turney et al. (2002) used 60 reviews for the movie 'The Matrix' and 60 reviews for 'Pearl Harbour'. He observed that appreciation of elements of a movie (events in it, actors in it), does not add up to appreciation of the whole movie, whereas appreciation of car parts will add up to appreciation of the car. One of his misclassified Pearl Harbour reviews contains the phrase "sick feeling", which has a negative semantic orientation, and the review was classified as 'thumbs down'. But the phrase refers to the event of the sudden bombing of Pearl Harbor, not to the movie, which was rated with five stars.

Turney et al. (2002) concluded that in movie reviews, the whole is not always the sum of the parts. This seems to be partly caused by the fact that the topic of the movie may interact with sentiments expressed in the review. In a documentary on some injustice, people may be expected to react with indignation and still recommend the documentary. What kind of topics may be present in movie reviews? We cannot know in advance. This motivates the idea of using latent semantic analysis (Deerwester et al., 1990), a technique that finds underlying concepts in a corpus of text in an unsupervised manner.

Latent semantic analysis (LSA) may also be used to calculate semantic orientation. Semantic orientation may be understood as semantic similarity. Turney & Littman (2003) compare LSA to PMI_IR in a task where words have to be classified as either positive or negative. To do this, their semantic similarity towards a set of positive and a set of negative words is computed in some corpus. They conclude that PMI_IR performs better because it can be calculated on a much larger corpus (the index of AltaVista). However, when evaluated on a corpus of comparable size LSA performs better. A key difference between PMI_IR and LSA is that PMI only uses cooccurrence of terms. LSA also associates terms with each other that hardly cooccur together, as long as they occur in similar contexts. Like in PCA, this is done by projecting term vectors on principal components. Landauer & Dumais (1997) interpret terms that are close together in the subspace spanned by the principal components as semantically related.

1.1.2 Supervised approaches

Pang et al. (2002) is another early work. They compare performance of several supervised classifiers on a total set of more than two thousand movie reviews, 759 labeled negative, and 1301 positive. Neutral reviews were left out. Support vector machines (Vapnik, 1982) combined with a unigram language model with a binary feature vector performed best at 82.9%. They improve on their own results in Pang & Lee (2004), where among other things they use a set of labeled positive and negative sentences during training. In this research they also introduce the dataset that we use: a dataset of two thousand labeled reviews, authored by over three hundred authors, with no more than twenty reviews per author.

Kennedy & Inkpen (2006) also work on this dataset with support vector machines. They, too, use support vector machines with binary unigram features with good results: 84.9%. They also experiment with a kind of semantic orientation calculated by term counting, but combining it with an SVM does not increase performance significantly. Only in combination with adding sophisticated bigrams that model negation and intensifiers using full parses of the review sentences can SVM performance be significantly improved, if only slightly.

Whitelaw et al. (2005) achieve the highest performance on this dataset known to us. They also model negation and intensifiers, but go further than that. They extract what they call "appraisal groups" from sentences. A lexicon of appraisal groups was first built in a semi automatic way. An appraisal group is modeled around an adjective, e.g. 'beautiful'. For each adjective in the lexicon, the 'attitude type' is given. This states if the adjective

- describes an emotional state of the writer (affect),
- expresses appreciation,
- or expresses a social judgement.

The adjective 'beatiful' expresses appreciation. Also, the semantic orientation of each adjective is given, 'beautiful' is of course positive. Then, with this lexicon they locate possible appraisal groups in reviews. By parsing the sentence around the adjective they allow negation to reverse the orientation. Then they count the number of appraisal groups with a certain attitude type and orientation, for each possible combination of these two properties. This leads to six features. Their best result was achieved by adding these features to bag of words features and using a support vector machine with a linear kernel: 90.2%.

Both Pang et al. (2002) and Kennedy & Inkpen (2006) used the SVMlight (Joachims, 2002) implementation of the support vector machine algorithm. They called SVMlight with all parameters set to their default value. In this research, we explore if varying parameters helps, such as normalizing the data in different ways and tuning the cost parameter. We also use the transductive support vector algorithm (tSVM) of SVMlight. In transductive machine learning, the algorithm may see the test points, but not their labels.

1.1.3 Combining LSA, semantic orientation and SVMs

LSA is very closely related to prinicipal component analysis (PCA). Looking ahead, our results in this research show that its use is limited for classification. This is a well known fact (see, e.g. Sun et al. (2004)), because PCA finds factors that describe the most of the variance in the data. These are not necessarily the factors that are most discriminative for classification. They are interesting objects of study by themselves. The factors that capture most variance are thought of as concepts in literature (Landauer & Dumais, 1997). Perhaps adding the top concepts as features to the original feature vector might help? In our theoretical background chapters, we already develop an intuition that this is not going to be easy.

Several approaches to make LSA supervised have been proposed (Sun et al., 2004; Chakraborti et al., 2007). These methods have in common that they use the class labels of training points. Our main interest was not so much in making LSA aware of the class labels, but rather to use it for interesting features. LSA similarity promises to capture semantic similarity. This allows us to define "interesting points" in the subspace spanned by the concept factors, the subspace where the reviews will be projected into. An interesting point could be a set of terms such as {exciting, excited, lively, enthousiastic, enthusiasm, elated, energetic, uplifting, fascinating}. This is a document, and it may also be projected on the LSA concept space by a process called "folding in". The LSA similarity of a review with this document might be an interesting feature for a support vector machine.

Instead of dreaming up a hand crafted lexicon of words that should be important in movie review classification, it is also possible to use an existing lexicon. Turney & Littman (2003) and Kennedy & Inkpen (2006) use the General Inquirer (GI) lexicon (Stone et al., 1966). This lexicon consists of 181 categories. Each category is a set of words that is related in the framework of some theory. Over several decennia, content analysts, psychologists and sociologists have contributed categories. Example categories are 'Positiv', 'Negativ', 'Active', 'Passive', 'Strong', 'Weak', 'Hostile'. Together, the categories are an interesting source of knowledge about language.

1.2 Research questions

Since Pang et al. (2002) and Kennedy & Inkpen (2006) use SVMlight only with its default parameters, a first question is: can classifier accuracy on the task of movie review classification be improved by tuning the parameters? Specifically, we will tune the cost parameter that is used in soft margin support vector machines (Cortes & Vapnik, 1995).

HTML pages can be quite noisy. They contain irrelevant headers and footers, links to movie review sites, and so on. We use suffix arrays (Manber & Myers, 1990) to remove noise in a semi-automatic way. Joachims et al. (1998) claims that SVMs can handle irrelevant features very well. Can we confirm that performance does not degrade if we do not remove any noise at all?

Second, does transductive machine learning improve accuracy? In the use case in our introduction where we downloaded documents related to the PVV of Geert Wilders, we are interested in the sentiment in a given set of documents.

This is an ideal setting for a transductive machine learning algorithm that can make use of test points. A related question is: is it possible to reliably estimate the sentiment of test points when only a limited amount of training data is available?

Latent semantic analysis finds factors that capture most variance. We find these factors for a movie review corpus of 2700+ documents compiled by Pang & Lee (2004). We also find them for just the two thousand labeled reviews. How does projecting the reviews on the subspace spanned by the orthogonal LSA factors affect accuracy? Are the concepts from the large corpus better in any way? In our theoretical chapters, we note the equivalence between LSA and PCA if the feature vectors are mean centered in advance. How does mean centering the data affect accuracy? We hypothesize that it does not matter because although the first principal component must point through the mean if data was not mean centered, the remaining principal components have many degrees of freedom left. Finally, does the amount of principal components that are kept matter? This value is interpreted to represent a property of the human brain by Landauer & Dumais (1997), where LSA is presented as a model for human learning. Results in literature vary, see Bradford (2008) for an overview.

We also explore possibilities of combining features obtained by latent semantic analysis to the unigram feature vectors. The first possibility we try is to add the first k principal components to the original feature vector. Another approach is to fold in all 181 GI categories, and calculate LSA similarities between the projected reviews and these “points of interest”. We first test the GI features obtained in this way by themselves. If they show promise, we examine if we can add them to the unigram feature vectors.

Chapter 2

Machine learning and Support Vector Machines

A scientific theory can often be formulated as a prediction. For example, using the second law of Newton, we may predict how long it takes for a falling ball to reach the earth. We take a number of measurements, such as the ball's starting position, its starting speed, and so on. We may choose to ignore certain properties of the ball, such as its diameter, or its mass. Our measurements then represent a simplified version of the falling ball. The second law of Newton may be thought of as a simplified model of the world. It takes the measurements as input, and outputs a prediction of the time it takes for the ball to touch down. Whether or not Newton's laws are 'true' is a matter of debate. But commonly, theories are regarded as better if they give 'better' predictions, e.g. they are on average closer to the observed time it takes for the ball to fall. This is a pragmatic approach to science.

model

Machine learning (Mitchell, 1997) can also be understood in this light. It is the name for a very broad research field that tries to find algorithms that can predict well. We always start with some observations. In this research these are movie reviews. We take some measurements, reducing the observations to simplified mathematical objects. In this research, we will roughly measure which words appear in the reviews. If we take n measurements, and any measurement can be represented by a real number, we may denote an observation as a vector $\vec{x} \in \mathbb{R}^n$. We say that $\mathbb{R}^n$ is our input space.

input space

The task of the machine learning algorithm is to learn a function that will accept measurements of other observations and outputs a prediction. In our study it will accept a description of a review and predict whether or not the review is positive. This is a special case of machine learning called pattern recognition or classification (Duda et al., 2001). Here the prediction to be made is just to which category an observation belongs. Another way to say this is that class labels have to be predicted. When there are only two classes, say a class labeled -1 and a class labeled 1, it is called a binary classification task. Then if $\mathbb{R}^n$ is our input space, we have to learn a function $g : \mathbb{R}^n \longrightarrow \{-1, 1\}$. This function is called the discriminant function.

pattern recognition

discriminant function

How can an algorithm learn a function? Normally we specify what kind of functions the algorithm can learn. Another way to say this is that we have to specify a set of functions that can be learned. In our research we will use a kind of linear classifier. A linear classifier can only learn a hyperplane. In two-dimensional space a hyperplane is just a line. A line separates the plane in two regions. These regions are called half spaces. Each region can be assigned to a class. In this way, a line can be used as a discriminant function. A line is determined by its slope and its intercept (the place where it intercepts one of the axes). These are the parameters of the line. A machine learning algorithm only has to learn the values of the parameters that yield the best predictions.

parameters

To estimate good values for the parameters, the algorithm needs to see some example observations. These observations form the training set. If during training the correct outputs for the example observations are known, a supervised algorithm can be used, otherwise an unsupervised algorithm has to be used. The performance of the fine tuned function is then evaluated on a set of observations for which the algorithm has not yet seen the predictions, the test set.

training set
supervised
unsupervised
test set

A distinction may be made between inductive machine learning and transductive machine learning (Vapnik, 1998). In inductive machine learning, we are interested in the mathematical model itself. The question is: can we use the learned model to make correct predictions about any possible new observation? If this proves to be the case, it is said that the model generalizes well. To answer this question, it is of paramount importance that the algorithm has never seen any properties of the observations in the test set. Only then may the observations in the test set be regarded as new.

induction
transduction
generalization

Related to the question of how well an algorithm generalizes is the problem of overfitting. Overfitting happens if an algorithm performs very well on the training set (it has a low training error), but it fails miserably on the test set (it has a high test error). We will see that this problem has a central role in the motivation for support vector machines, the main machine learning algorithm used in this research.

overfitting
training error
test error

In transductive machine learning, the aim is more modest. In a pattern recognition task, given a training set and a test set, we only ask: can we accurately predict the labels of the observations in the test set? Here we may estimate the parameters of our model using all observations. This includes the observations in the test set, as long as the algorithm does not see the labels of the test set instances. Because of this, it may be expected that the algorithm performs equally well or better on the test set than in the inductive setting. If in solving some real life problem, all data of interest is available, then the idea of transductive machine learning is: why not use it. Why solve the more difficult problem of predicting any unseen point, when all you want is to predict known instances? The advantage of transductive machine learning is small when relatively much training data is already available. If the training set is relatively small, however, using the test points may improve the quality of estimates of various kinds.

In the next few sections a few different approaches for the pattern recognition problem are discussed. First, so called parametric approaches will be treated, with the well known Naive Bayes algorithm as an example. Then, we will discuss some concepts of statistical learning theory. Support Vector Machines are the most widely known exponent of this approach.

2.1 Parametric approaches

The hope in pattern recognition is that we can find some features, some characteristics, of events or objects on which the category they belong to depends. If this is not the case, how could we ever hope to predict the correct category for an observation?

The first step in a parametric approach is to guess which distribution generated the obervations. This expression shows an interesting view of the world: every set of properties can be modeled by some multivariate probability distribution. When there are more than three or even thousands of properties, it is hard to visualize the data. Hence, it is hard to guess which distribution generated the data. This is a problem of the parametric approach. Often, the normal distribution is selected. It is such an attractive candidate because of the central limit theorem, which states that the sum of a large enough set of random variables drawn from whatever distributions follows a normal distribution. Thus, if many unknown factors are deemed to contribute to the magnitude of some quantity, this distribution is a first candidate for selection. Still, often enough random variables do not follow a normal distribution at all.

central limit theorem

In the pattern recognition setting, it is common to assume that points of each class were generated by a separate distribution. How does this work? First, a class is selected, each class c_i has a chance to be selected: $P(c_i)$. These probabilities are called the class priors, or just the priors. Then from the distribution that belongs to class c_i a data point is generated. This distribution can be written as $P(\vec{x}|c_i)$, which reads: the probability of observing $\vec{x}$ *given* that this observation has class label c_i. It is called the class conditional distribution. The overall density is then just the weighted summation over these class conditional distributions: $P(\vec{x}) = \sum_i P(c_i)P(\vec{x}|c_i)$. If the Gaussian distribution is selected for $P(\vec{x}|c_i)$, the overall distribution is called a mixture of Gaussians.

priors

class conditional distribution

The second step is to estimate the parameters of the class conditional distributions. A multivariate normal distribution over n variables has a mean $\mu \in \mathbb{R}^n$, and a covariance matrix $\Sigma \in \mathbb{R}_{n\times n}$. Note that this notation means that Σ belongs to the class of matrices that has n rows and n columns. The covariance matrix is symmetric by definition, hence it contains $\frac{1}{2}n(n+1)$ independent parameters. In total, for each class, $n + \frac{1}{2}n(n+1)$ parameters have to be estimated. The number of parameters in this case is quadratic in n, which constitutes another problem with the parametric approach if the data is high dimensional. Intuitively this is not hard to see. If one needs, say, 30 data points on a line to satisfactorily estimate the mean and variance of a normal distribution, then on a plane it is not strange if you require a couple hundred points. In a three dimensional volume you might already want more than ten thousand points. With high dimensional data one can see that the number of data points to reliably estimate a multivariate distribution becomes impossibly large.

Also, a model with many parameters can be too “powerful” if there is little data available. Such a model can take many shapes in the n-dimensional input space and this means that it might model the training instances too perfectly. Instances that are outliers of a simple model occurring by chance may be explained as more probable instances of a more complex model that less accurately captures the structure of the underlying data distribution. Then, if test instances are presented, the learned model may fail. This is the problem of overfitting that we already mentioned. A surprisingly robust solution to these problems is the subject of the next subsection.

How do we classify a new test instance once we have estimated the probability distribution? We ask which class conditional distribution has most likely generated the test point. We can write this for a test point $\vec{x}$ as:

$$P(c_i|\vec{x}) = \frac{P(\vec{x}|c_i)P(c_i)}{P(\vec{x})}.$$

This is called the Bayes rule (Duda et al., 2001).

2.1.1 Naive Bayes

The central assumption in the Naive Bayes (John & Langley, 1995) approach is that in the class conditional probability distribution, the features are independent. $P(\vec{x}|c_i) = \Pi_j P((\vec{x})_j|c_i)$, where $(\vec{x})_j$ is the j'th feature, which is a random variable. For the multivariate normal distribution, this assumption causes the covariance matrix to become diagonal, because independent random variables have zero covariance. Thus, now only the mean and the diagonal of the covariance matrix have to be estimated, a total of $2n$ parameters, linear in the number of dimensions. Thus, less data is necessary to make a reasonable estimation, and the danger of overfitting is reduced. Even though the density estimations may be poor, classification performance is very competitive in many problems (John & Langley, 1995), because for classification we only need to know which class was most likely to have generated a test point.

2.2 Statistical learning theory

Because of the problems associated with parametric approaches to pattern recognition problems, statistical learning theory was developed in the sixties and seventies (Vapnik, 1998). We noted already that in many cases a researcher may not know the underlying distributions that generated the observations of the classes. Even if he can make an educated guess, shortage of observations may make it very hard to reliably estimate the parameters.

Statistical learning theory was developed around the problem of binary classification. This made it possible to define the central concepts, one of which we describe below, in an elegant way. The theory was later generalized to other types of statistical inference (Vapnik, 1998). It aims to formalize some aspects of learning algorithms that work independently of the underlying data distributions.

Two concepts are central to this approach. First, the idea that minimizing the error on the training set is important. This is obvious and it is also the idea behind parametric approaches. If we already make many mistakes on the training set, how can we expect to perform well on the test set? Second, we have to beware of overfitting, we do not want to model the training set too perfectly. Vapnik & Chervonenkis (1968) introduced a very elegant concept to characterize the "power", or capacity of a learning algorithm: the Vapnik-Chervonenkis dimension, more often referred to as the VC-dimension. We have noted already that the more powerful a learner, the more prone to overfitting it is. The idea then, is to minimize the training error with a model that has a VC-dimension as small as possible.

capacity
VC-dimension

The definition of the VC-dimension of a binary classifier is remarkably simple: it is the maximum number of points that the algorithm can still shatter. **shatter** An algorithm can shatter a set of points if its parameters can be adjusted during learning in such a way that it can divide the points in two sets in any possible way. In other words, it can always achieve zero training error on this data set.

A simple example to illustrate the concept of VC-dimension is a linear classifier. Suppose that the instances are two-dimensional, then a line is learned. The reader may verify that any three points that do not lie on the same line can be divided into two sets in any possible way (although to prove that the VC-dimension is at least three it would suffice to find just one set of three points that can be shattered). Four points can not be shattered anymore by a line. No matter how they lie in the space, it is always possible to assign them labels such that it is not possible to draw a line between them that classifies all four correctly (Try it!). An example is the famous XOR problem. If the instances are n-dimensional, a linear classifier learns a hyperplane. It is not hard to prove that a hyperplane in an n-dimensional space can shatter at most $n + 1$ points, see (Burges, 1998). Thus, the VC-dimension of linear classifiers is linear in the number of dimensions.

Philosophically, the simple concept of a VC-dimension is interesting. Vapnik (1998) relates it to both Occam's razor and Karl Popper his ideas about falsification. Occam's razor is often quoted as stating "the simplest explanation is the best". Statistical learning theory states that the simplest classifier is the one with the lowest VC-dimension. Karl Popper famously claims that a scientific theory is falsiable. A classifier with a low VC-dimension can be falsified by a problem with few data points, so it would qualify as a scientific theory according to Popper.

(Burges, 1998) is an excellent and entertaining tutorial in which the interplay between VC-dimension, sample size, training error and test error is treated in depth. It discusses the striking result from statistical learning theory that with a certain probability, the test error has an upper bound that is determined by the training error, the sample size of the training set, and the VC-dimension of the classifier. The lower the training error and the VC-dimension and the higher the sample size, the lower the bound. This bound on the risk of misclassification is independent of the probability distributions of the classes.

This is all great, but Burges (1998) also cautions the reader not to disregard algorithms with infinite VC-dimension. Even though statistical learning theory in such a case does not give an upper bound on the test error, such algorithms can still perform well. An example is the k-nearest neighbour algorithm. With k=1, it will score 100% on any training set with any labeling (simply assigning the label of the training instance to itself). Thus, it has infinite VC-dimension. Still, in practice, it often performs well. Interestingly, for the informed reader, support vector machines with a radial basis function as a kernel also have infinite VC-dimension.

2.3 Support Vector Machines

Support vector machines are a special kind of linear classifiers. They learn a hyperplane, but not just any hyperplane that separates the training points correctly. If the training points are linearly separable, we can draw a convex hull **convex hull**

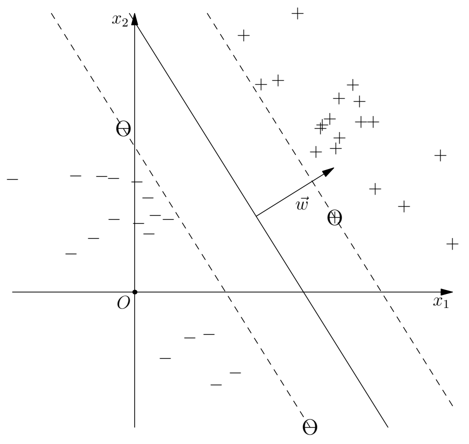


Figure 2.1: The maximum margin hyperplane found by the SVM implementation of SVMLight (Joachims, 1999a) on a toy training data set. The support vectors are encircled.

around the instances of the two classes. A shape is convex if for any two points inside it, all points on the line between them are also inside the shape. The two points (one on both of the hulls) where the convex hulls are the closest may be connected by a line. The hyperplane perpendicular to and crossing the middle of this line is the plane that a support vector machine learns. In this way, the points of the different classes that are closest together -the most difficult points- are as far away as possible from the hyperplane. In other words, the classifier maximizes the margin between the closest training points. Therefore, SVMs are also called maximum margin classifiers. The points that lie on the margin of the widest possible hyperplane are called the support vectors.

maximum margin

In Figure 2.1 we see a toy classification problem. The minuses were generated with a bivariate Gaussian with mean $\mu = (0.25, 0.5)$ and covariance matrix $\Sigma = \begin{pmatrix} 0.75 & -0.75 \\ -0.75 & 1.25 \end{pmatrix}$, the pluses with $\mu = (3, 2)$ and $\Sigma = \begin{pmatrix} 0.75 & -0.75 \\ -0.75 & 1.25 \end{pmatrix}$. The class priors are equal, that is there are as many pluses as there are minuses. Using all the plotted data points as a training set, an SVM will find the hyperplane as plotted.

We see that the support vectors (encircled in the picture) completely determine the hyperplane and its margin. The position of the other data points is irrelevant. Indeed, SVMs are not interested in describing the probability distributions that generated the data. Their sole purpose is to minimize the risk of misclassifying new points drawn from them. Statistical learning theory states that this can be achieved by two things, minimizing the error on the training set, and minimizing the VC dimension of the learned hyperplane.

If the training instances for both classes are linearly separable, obviously a support vector machine finding the maximum margin hyperplane will achieve a perfect score for the training set. If the training instances are not linearly separable, slack variables are used. The idea is that now some training points are allowed to be on the wrong side of the hyperplane. Also in this case the idea remains that the error on the training set must be minimized. Usually, the sum

slack variables

of the Euclidean distances between the hyperplane and the misclassified points are used as an indication of the severity of the error, so this has to be minimized by the plane.

Minimizing the VC-dimension is done by maximizing the margin. Burges (1998) notes that there is no rigorous proof available yet to determine the VC-dimension of support vector machines, there are only plausibility arguments. Intuitively, the larger the margin of a hyperplane, the less "wiggle room" it has. Even though a "fat" line in two-dimensional space still shatters three points, all these points have to be separated from each other by at least the margin around the line. Given that the points come from some probability distribution, the larger the margin, the less likely it is that the plane shatters the points. This is just a personal intuition, but for more in depth arguments, see (Burges, 1998), or (Vapnik, 1998).

Joachims et al. (1998) showed that in practice SVMs do not suffer from overfitting in the task of text categorization, which is a classification problem where the categories are topics, such as news categories from press agencies. As in our research, his data points were high dimensional with 9962 features. It is even possible to use many more features, by also using combinations (such as products) of two or more features as additional features. This resilience against very high dimensional data points is what makes SVMs one of the machine learning algorithms that lends itself for use of the kernel trick. We will explain the kernel trick below, but for now note that the addition of products of two or more features can be achieved using a polynomial kernel of degree two or more. Thus, a polynomial kernel may be used if one expects products of individual features to contain interesting information for classification.

text categorization

kernel trick

2.4 Finding the maximum margin hyperplane

In this section, we will give some geometrical properties of the maximum margin hyperplane that an SVM finds on a dataset, and we will introduce some notation that will be helpful when we look at how this hyperplane is found. The reader may find it helpful to look at Figure 2.1 while reading the next few paragraphs.

Let $D = \{\vec{x}_1, \vec{x}_2 \ldots | \vec{x}_i \in \mathbb{R}^n\}$ be a set of datapoints, with corresponding labels $y_i \in \{-1, 1\}$. A hyperplane is itself a space that is $n-1$ dimensional. So there is one direction (dimension) "missing" from the space. This is the direction normal or orthogonal to the space, this is the direction of $\vec{w}$.

normal

Let $\vec{w}$ be a vector in this direction. Then, for all the points $\vec{x}$ on the hyperplane it holds that the dot product of them with w must be equal to some constant $(-b) \in \mathbb{R}$, so we can write the equation for the hyperplane as

$$\vec{w} \cdot \vec{x} + b = 0$$

It is then easy to see that the distance of the hyperplane to the origin is given by $\frac{|b|}{||w||}$ For our research, we are mainly interested in the direction of $\vec{w}$. Why? Because this is the normal to the hyperplane. It is the only direction that matters in deciding to which class a data point belongs! Therefore, features (dimensions) that have a larger absolute value in $\vec{w}$ are seen as more important or more relevant for classification by the SVM. We will use this later to find lists of important review words for reviews of movies from different genres. It is important to note, however, that $\vec{w}$ is influenced by the scaling of the input axes. For instance, if one of the axes was measured in centimeters, and the other in meters, $\vec{w}$ might be much more parallel to the centimeter axis. The direction of $\vec{w}$ is also affected by preprocessing steps such as standardization or normalization.

level set

Let $g(\vec{x}) = \vec{w} \cdot \vec{x} + b$, then its level sets, the sets of points in its domain where $g(\vec{x}) = c$ for some constant c, are all parallel to the hyperplane $g(\vec{x}) = 0$. On one side of this hyperplane it holds that $g(\vec{x}) < 0$, on the other side $g(\vec{x}) > 0$. Once we found the optimal hyperplane, we use $g(\vec{x})$ as a discriminant function. We give it a test point, and if it gives a positive value, we assign it the label of the training points on the positive side of the hyperplane (the label 1). If it gives a negative value, we give it the label of the training points on the negative side of the plane (the label -1).

The Euclidean distance between a test point $\vec{x}'$ and the plane $g(\vec{x}) = 0$ is given by $\frac{|g(\vec{x}')|}{||w||}$. This value can be used to give an indication of how strongly the SVM believes the point to belong to its class: the larger the distance, the stronger the confidence. Now consider the level sets $g(\vec{x}) = \pm 1$, the dashed lines in Figure 2.1. By minimizing $||\vec{w}||$, we can push them further away from the solid line in the middle. But we may not push them further away than any of the training points, so for all i it must hold that $|g(\vec{x}_i)| \geq 1$, or $y_i g(\vec{x}_i) \geq 1$, using the class labels $y_i \in \{-1, 1\}$.

$||\vec{w}||$ is a little awkward to minimize, since if you write it out, you get a square root. Luckily, its minima coincide with the minima of $\frac{1}{2}||\vec{w}||^2$. If we minimize this subject to the above constraints, we have a standard quadratic programming problem: Minimize with $\vec{w}$ and b

quadratic programming

$$\frac{1}{2}||\vec{w}||^2$$

subject to the constraints

$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0$$

for all i.

objective function

The function to be minimized is called the objective function.

2.5 Quadratic programming, a simple example

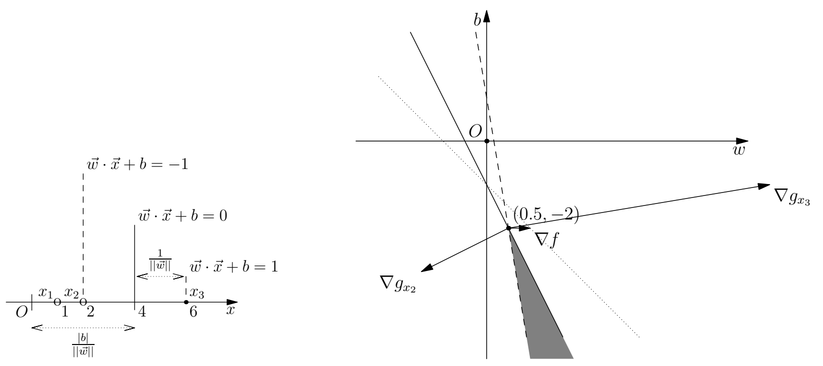
In Figure 2.2a, a one dimensional data set is shown. which is linearly separable. Obviously, the optimal hyperplane is located at the point $x = 4$ right in the middle of the two closest points $x_2 = 2$ and $x_3 = 6$, with class labels $y_1 = -1$ and $y_2 = 1$, respectively. These points are the support vectors. The point $x_1 = 1$ is irrelevant to the position of the hyperplane.

The quadratic programming problem for this case simplifies to: Minimize with w and b:

$$\frac{1}{2}w^2$$

subject to the constraints

$$y_i(wx_i + b) - 1 \geq 0$$



(a) A one dimensional data set

(b) The feasible region is the dark triangle

Figure 2.2: A one dimensional data set and its quadratic programming problem, in which the reader can verify that $w = (0.5, -2)$ is a correct solution and that $\nabla f = \lambda_2 \nabla g_2 + \lambda_3 \nabla g_3$ for $\lambda_2 = \lambda_3 = \frac{1}{8}$.

for all i.

We can plot w against b, and then plot the constraint given by each training point in there, see Figure 2.2b. The constraints are lines. If we write out the constraints, we get:

$$\begin{aligned}
g_1(w,b) = y_1(w * x_1 + b) - 1 = -w - b - 1 \geq 0, &\text{ the dotted line}\\
g_2(w,b) = -2w - b - 1 \geq 0, &\text{ the solid line, and}\\
g_3(w,b) = 6w + b - 1 \geq 0, &\text{ the dashed line}
\end{aligned}$$

in Figure 2.2b.

Each constraint is a hyperplane (in this case, a line). On one side of the hyperplane the inequality holds, on the other side it does not. Trying some point, for example the origin, for each constraint gives a feasible region, in which all constraints hold. A quick glance at the constraints shows that none of them holds in the origin. Thus, the origin lies in the infeasible region for each of the constraints. This leaves the small grey triangle as the feasible region in which we search for the optimal value of f. We can see immediately that constraint g_1, given by point x_1 indeed plays no role in defining the feasible region: its infeasible region lies entirely in the union of the infeasible regions of the constraints of the support vectors. **feasible region**

The feasible region is convex, which is always the case when the constraints are linear functions. f, the objective function, is also convex and it is bounded below in the feasible region. It obtains the smallest value on a vertex of the feasible region, $(0.5, -2)$. Solutions to this type of problems always lie on vertices, and because of convexity of f, an easy method to find the optimal value is to start at a vertex of the feasible region, and walk from there to a neighbouring vertex where f obtains a lower value, until no such neighbour is at hand. This is called the simplex method. **simplex method**

However, in most treatments of support vector machines, we find another approach, which makes use of a so called dual quadratic programming problem. This alternative statement of the problem allows for faster solving algorithms, and it also allows the application of the already mentioned kernel trick. To introduce the intuition behind this, it is easier to start with a quadratic programming problem in which we have to find extreme values of a function $f(\vec{x})$ subject to a constraint $g(\vec{x}) = 0$. This just means that we can only consider values in the domain of f that are in the level set with value 0 of g. Then the Lagrange theorem (Marsden & Tromba, 2003), page 226, states that at local extreme values $\vec{x_0}$ of f in this restricted domain,

$$\nabla f(\vec{x_0}) = \lambda \nabla g(\vec{x_0}), \lambda \in \mathbb{R}$$

gradient

where $\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_i} \\ \dots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$ is the gradient of f, which is the direction in its domain for which it changes fastest. This is not hard to understand, a nice example can be found on the Wikipedia page on Lagrange multipliers: suppose that $\vec{x} \in \mathbb{R}^2$, and $g(\vec{x}) = 0$ is some curve in that plane. Walking along that curve, at all times the direction in the domain of g in which it changes fastest is perpendicular to the direction of the level curve. If we now arrive at an extreme value of f, then in the direction of the curve, f, too, does not change. The direction of fastest change is also perpendicular to the level curve of g. λ is called a Lagrange multiplier.

If we have multiple constraints $g_1(\vec{x}) = 0, g_2(\vec{x}) = 0, \dots$, then it holds that the direction of fastest change of f in all extreme values of f restricted to the values in the intersection of the level sets of the g_i functions must be a linear combination of the gradients of the constraints in these points:

$$\nabla f(\vec{x_0}) = \sum_i \lambda_i \nabla g_i(\vec{x_0}).$$

Rewriting the above, we have:

$$\nabla f(\vec{x_0}) - \sum_i \lambda_i \nabla g_i(\vec{x_0}) = 0. \qquad (2.1)$$

On the Wikipedia page on Lagrange functions, it is hypothesized that at this point Lagrange must have noticed that this equation resembles the solution of setting to zero the derivative of a function $L(\vec{x}, \vec{\lambda})$:

$$\nabla L(\vec{x}, \vec{\lambda}) = 0,$$

where

$$L(\vec{x}, \vec{\lambda}) = f(\vec{x}) - \sum_i \lambda_i g_i(\vec{x}).$$

The function L is called the Lagrange function. Setting the gradient to the zero vector gives a system of linear equations:

$$\frac{\partial L}{\partial x_1} = \frac{\partial f}{\partial x_1} - \sum_i \lambda_i \frac{\partial g_i}{\partial x_1} = 0$$

$$\dots$$

$$\frac{\partial L}{\partial \lambda_1} = g_1(\vec{x}) = 0$$

$$\dots$$

We can see that this indeed corresponds to equation 2.1 and the constraints.

Let us now pause a little and take a look at Figure 2.2b, to make this a little more concrete and to see that we get good results with this calculation. We see that the solution lies at the intersection of the two constraints $g_2(w, b) = -2w - b - 1 = 0$ and $g_2(w, b) = 6w + b - 1 = 0$. If both of these functions equal zero, so does their sum, which leads to $4w - 2 = 0$ which leads to $w = 0.5$ and $b = -2$. We have just solved a very simple system of linear equations, which we derived above from the Lagrange function. In doing this, we have found the optimal hyperplane to our one dimensional example.

The Lagrange function for this problem is $L(w, b, \lambda_2, \lambda_3)$. Note that we discard the constraint given by datapoint $x_1 = 1$, since its infeasible region is entirely inside the infeasible region of the point $x_2 = 2$. So we assume that our algorithm has already determined the feasible region here. Setting the gradient for the Lagrange function to zero also enables us to find the Lagrange multipliers:

$$\frac{\partial L}{\partial \vec{w}} = w - \lambda_2(-2) - \lambda_3 6 = 0.5 + 2\lambda_2 - 6\lambda_3 = 0$$
$$\frac{\partial L}{\partial b} = 0 - \lambda_2(-1) - \lambda_3 = \lambda_2 - \lambda_3 = 0$$

We have substituted the solution we found for w and b here and this small system of linear equations gives us $\lambda_2 = \lambda_3 = \frac{1}{8}$.

2.6 Duality of the quadratic programming problem

Instead of searching for the feasible region defined by the constraints and then searching for the vertex that contains the solution, it is also possible to work directly from the Lagrange function. If D is our set of inequality constraints , then

$$L(\vec{x}, \vec{\lambda}) = f(\vec{x}) - \sum_{i \in D} \lambda_i g_i(\vec{x})$$

is the Lagrange function. Recall that we have to minimize f, subject to the constraints $g_i(\vec{x}) \geq 0$. This time, all the constraints are still in the Lagrange function. We have not yet found the feasible region, let alone the two constraints that meet at the vertex of the feasible region that contains the solution. Above we outlined how we could do this, and it would correspond to setting to zero all the λ values that do not define our feasible region.

In Figure 2.2b we see that the gradient of f points in between the gradients of g_2 and g_3. This is a standard property of quadratic programming, if the gradient would point outside of the constraints, the optimal solution would be at another vertex of the feasible region. It entails that $\nabla f = \lambda_2 g_2 + \lambda_3 g_3$ for *positive* λ_2 and λ_3. Thus, in the above Lagrange function we require:

$$\lambda_i \geq 0 \ \forall i. \tag{2.2}$$

Duality theory now is about switching the problem around. In some cases this is possible, and the SVM formulation is one of these cases. An important property of it is that both the objective function and the feasible region are convex. One of the ways of turning this problem upside down is the Wolfe dual. Instead of minimizing L with $\vec{x}$ subject to the constraints that all λ-terms vanish, we can *maximize* L subject to the constraints that

Wolfe dual

$$\frac{\partial L}{\partial \vec{x}} = 0. \tag{2.3}$$

Intuitively, this means that we require f to have an extreme value. But some of these may lie outside the feasible region. To get to the edge of the feasible region, we must now maximize with $\vec{\lambda}$. The study of exactly under which conditions this works is the subject of duality theory. We keep the requirement that $\lambda_i \geq 0$

Now let us see where this brings us with the SVM problem formulation. The Lagrange becomes:

$$L(\vec{w}, b, \vec{\lambda}) = \frac{1}{2}||\vec{w}||^2 - \sum_{i \in D} \lambda_i \left(y_i(\vec{w} \cdot \vec{x_i} + b) - 1 \right), \tag{2.4}$$

which we can rewrite to:

$$\frac{1}{2}||\vec{w}||^2 - \sum_{i \in D} \lambda_y y_i \vec{x} \cdot \vec{w} - b \sum_{i \in D} \lambda_i y_i + \sum_{i \in D} \lambda_i. \tag{2.5}$$

The Wolfe conditions are then:

$$\frac{\partial L}{\partial \vec{w}} = 0 \Longrightarrow \vec{w} = \sum_{i \in D} \lambda_i y_i \vec{x_i}, \tag{2.6}$$

and

$$\frac{\partial L}{\partial b} = 0 \Longrightarrow -\sum_{i \in D} \lambda_i y_i = 0. \tag{2.7}$$

Equation 2.7 tells us that in equation 2.5 the term that contains b drops out immediately. With equation 2.6 we can now write L succinctly as:

$$L = \frac{1}{2}\vec{w} \cdot \vec{w} - \vec{w} \cdot \vec{w} + \sum_{i \in D} \lambda_i = -\frac{1}{2}\vec{w} \cdot \vec{w} + \sum_{i \in D} \lambda_i \tag{2.8}$$

Using equation 2.6 again we can write out $\vec{w} \cdot \vec{w}$ and obtain

$$L = -\frac{1}{2} \sum_{i,j \in D} \lambda_i \lambda_j y_i y_j \vec{x_i} \cdot \vec{x_j} + \sum_{i \in D} \lambda_i \tag{2.9}$$

We can now maximize this function with $\vec{\lambda}$ subject to the constraints:

$$\lambda_i \geq 0 \quad \forall i. \tag{2.10}$$

The feasible region in this dual quadratic programming problem is much easier to handle. We can now put $\frac{\partial L}{\partial \vec{\lambda}} = 0$ to find the maximum. In our one dimensional example of figure 2.2b, this should lead to a system of three equations with the solution $\lambda_1 = 0$, $\lambda_2 = \lambda_3 = \frac{1}{8}$. The constraints, the positive lambda constraint (equation 2.2), the condition that a solution has to be an extremum of the Lagrangian with regard to the objective function variables (equation 2.3) and the condition that all lambdas of 'irrelevant' constraints are set to zero are together called the Karush-Kuhn-Tucker (KKT) conditions. The last condition can be more elegantly stated as

$$\lambda_i(y_i(\vec{w} \cdot \vec{x_i} + b) - 1) = 0, \forall i \in D \tag{2.11}$$

The solution of the dual gives us $\vec{\lambda}$, and with equation 2.6 we can obtain $\vec{w}$. The above equation gives us a way to get b. For the support vectors, λ_i is nonzero, so $y_i(\vec{w} \cdot \vec{x_i} + b) - 1 = 0$ and $b = -\vec{w} \cdot \vec{x_i} + y_i$. Taking the average b for all support vectors is considered the best way to obtain b.

Besides it being easier to solve, another advantage of this dual formulation of the problem is that the data points in our training set only appear inside a dot product. This is what allows the famous kernel trick. An often heard misunderstanding is that support vector machines project datapoints on a higher dimensional space. By now it should be clear that support vector machines do nothing of the sort: they just find a wide hyperplane that separates the points. It is the kernel trick that simulates a projection to a higher dimensional space. This trick can be used in any algorithm in which only the dot product between data points is used.

kernel trick

What is the trick then? We can substitute the dot product with another function $K(x_i, x_j)$. If this function fulfills certain conditions (the Mercer conditions) then it corresponds to the dot product of the datapoints in a higher dimensional space. This space is often referred to as the feature space. Because SVMs work well with high dimensional data the kernel trick is often used with them. It is useful if the data in the input space is not linearly separable, because it can be proven that by projecting a dataset to some suitable higher dimensional space, any dataset becomes linearly separable. Which kernel function to use? This is in fact an area where the researcher has much freedom. In fact Burges (1998) calls the choice of kernel “a very big rug to sweep parameters under”.

feature space

2.7 Slack variables improve generalization

We already discussed how slack variables may be used to handle cases where training data points are not linearly separable and how it would make sense to minimize the summed Euclidean distance between misclassied points and the hyperplane. Then a tradeoff has to be made between the width of the hyperplane and the error. We will see below that this is done by introducing a ‘cost’ parameter C. This parameter has to be chosen beforehand to allow the SVM to find the optimal hyperplane. To optimize this parameter for a given problem one is therefore obliged to use the standard techniques. One particularly simple approach is to try several values and use the best, this is commonly referred to as grid search (Hsu et al., 2003), because if multiple parameters would have to be optimized this way, you would just try each point on the grid spanned by the possible values of the parameters.

grid search

Introducing slack variables $\xi_i \geq 0$ for each datapoint $\vec{x_i}$ we can rewrite the constraints as:

$$\vec{w} \cdot \vec{x_i} + b \geq 1 - \xi_i$$

for points labeled with $y_i = 1$ and

$$\vec{w} \cdot \vec{x_i} + b \leq -1 + \xi_i$$

for negatively labeled points, and minimize $\frac{1}{2}||\vec{w}||^2 + C\sum_{i \in D} \xi_i$. It can be shown that this leads to exactly the same dual as in equation 2.9, but now subject to the constraints:

$$0 \leq \lambda_i \leq C \ \forall i.$$

Interestingly, even if training points are linearly separable, this could be by chance. By allowing some error on the training points, the margin of the hyperplane can grow larger. Intuitively, this decreases the VC-dimension and improves generalization. Consider also that any non linearly separable set can be made separable by projecting the data points on some higher dimensional feature space. This does increase dimensionality, however, and introducing slack variables is then a good idea to increase the margin of the hyperplane.

2.8 Transductive Support Vector Machines

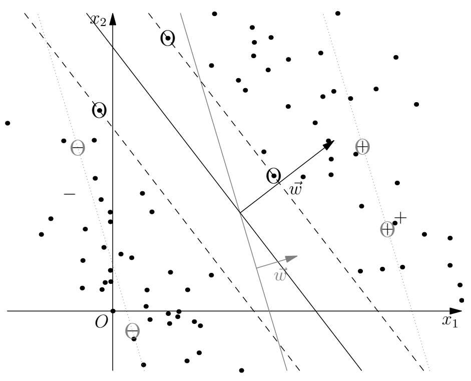


Figure 2.3: The maximum margin hyperplane found by the transductive support vector machine by Joachims (1999b), on a toy data set generated from the same bivariate Gaussian that was used in Figure 2.1. The pluses and minuses are the only labeled instances. In grey, the maximum margin hyperplane for these points is plotted, as are the encirled support vectors for that plane. In black, maximum margin hyperplane for all the points is plotted, as are the encircled support vectors for that plane.

Joachims (1999b) introduced an algorithm for transductive learning with support vector machines. In transductive machine learning, besides the training points in D, we also use test instances $\vec{x_i}' \in D'$. See Figure 2.3 for an example of a hyperplane found with this algorithm, which is implemented in SVMLight, software developed by Joachims, freely available for research purposes.

The challenge is to find a labeling $(\vec{x_i}', y_i') \ \forall i \in D'$ and a hyperplane $\vec{w} \cdot \vec{x} + b = 0$ such that the margin of the hyperplane is maximized. Below we write this labeling as a vector $\vec{y}' \in \{-1, 1\}^{|D'|}$.

In Figure 2.3 we see how the transductive algorithm works. Only six labeled data points are available. The first step is to find the maximum margin hyperplane for the labeled instances. It is depicted in grey in the figure. Its support vectors are encircled in grey. In the second step, it estimates the class priors from the training set. These are the probabilities that a test set point belongs to the negative or the positive class, respectively. It is also possible to tell SVMlight explicitly how many points in the entire set (the union of the training and the test set) are positive with a command line parameter. In this research, we did not do this, however, because in a real life situation the class priors are often unknown.

Based on the estimated class prior for the positive class, SVMlight calculates how many points in the entire set it will assign the positive class label. Suppose these are p points. Now, the p instances furthest from the hyperplane on the positive side are assigned to the positive class. After that, the algorithm enters a loop, in which it swaps the labels of test instances such that the margin of the hyperplane is increased. The solution it finds is an approximation. If the class prior estimation is off, during training the wrong number of test points will be assigned to the positive class. While it searches for the maximum margin hyperplane it only swaps class labels, so p never changes. During testing, this will then result in errors. If only a very limited amount of training data is available, the estimation may be off, but in Figure 2.3, they are just right.

When slack is allowed, the unlabeled instances also each receive a slack variable ξ'_i. A second cost parameter C' can be set by the researcher to manipulate the tradeoff that the algorithm makes between the error on unlabeled instances (at each point in the algorithm, each instance has an assumed label) and the margin of the hyperplane. All of this leads to the following Lagrange function: Minimize with $\vec{w}, b, \vec{y}', \vec{\xi}, \vec{\xi}'$:

$$\frac{1}{2}||\vec{w}||^2 + C \sum_{i \in D} \xi_i + C' \sum i \in D' \xi'_i$$

subject to the constraints:

$$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 - \xi_i,\ \xi_i > 0\ \forall i \in D$$
$$y'_i(\vec{w} \cdot \vec{x_i}' + b) \geq 1 - \xi'_i,\ \xi'_i > 0\ \forall i \in D'$$

Vapnik (1998) treats the problem of transductive learning with support vector machines already, but Joachims (1999b) made it feasible for the typically high dimensional problems encountered in natural language processing tasks, because his algorithm converges quickly. We have repeated the formal statement of the problem here to show that it is quite complicated, and many variables have to be considered in the optimization. A global optimum could be reached by trying all assignments for the unlabeled instances, but these increase exponentially with the test set size. That is why SVMLight starts with the reasonable hyperplane that is obtained by using just the labeled instances. From there, by swapping labels of test points, a local search is carried out that tries to solve the above optimization problem, even if the eventual solution might be a local optimum.

2.9 Normalizing the input data

Hsu et al. (2003) note that scaling or normalizing the input features is very important for support vector machine performance. They note that for neural networks normalization is important and state that most considerations also apply for support vector machines. At first sight, this may seem suprising. Does a hard margin SVM not find a global optimum for the separating hyperplane? If one scales the input axes, would it not just find exactly the same hyperplane, rotated to accomodate the scaling? If one translates the data, would the support vector machine not simply adjust the bias of the hyperplane?

First, the fact that we are using a soft margin support vector machine complicates the matter. While for a given value of the hyperparameter C it still finds a global optimum, we don't try every possible value for C. For example, it may be that the default value that SVMlight calculates for C when no value is specified hits the mark better in a normalized input space. And the transductive algorithm implemented in SVMlight does not find a global optimum at all. It should be expected that normalization can affect its performance.

Second, some normalizations change feature vectors in ways that alter the information. In the subsections below we shortly discuss some forms of normalization or preprocessing of the feature vectors.

2.9.1 Term frequencies or binary features

In the term by document matrix, term frequencies are the features. With binary features we mean that we do not take into account how often a word appears in a document, only if it does occur or not. Landauer & Dumais (1997) uses a logarithm to dampen the term frequency of words. They defend it in their model of human learning saying that the learning effect is strongest for the first occurrence of a word in a document, and declines with repetitions. Binary features are even more extreme, only the first occurrence is taken into account. Drucker et al. (1999) note that for support vector machines binary features worked best, compared to tf-idf and term frequency features.

Binary features create an interesting space. Only the vertices of a unit hypercube are occupied. If there are two or more points located at the same vertex, this is likely to be a duplicate review (even though two reviews with the same words could contain them in a different order).

2.9.2 Normalizing with sample mean and sample standard deviation

Here, we first locate the sample mean document vector, and then we take this as the origin of the space. Or, equivalently, we subtract from each feature its mean over all document vectors. The mean of a series of values is sensitive to outliers. If the distribution of the values of a feature is unimodal, the mean can be interpreted as the prototypical value. If the distribution is multimodal, the mean is in itself not a very useful value. Subtracting it from all values will always center the points around the origin.

As a second step, the axes are all scaled, they are divided by the sample standard deviation of the corresponding features. If some feature shows no variance at all, the corresponding axis would be scaled to infinite length. If this happens, we choose not to scale the axis at all. This makes sense because such a feature can hardly be discriminative for classification.

If the values of a feature are from a normal distribution, then this normalization centers gives a t-distribution centered around the origin. But even if the values are from a very different distribution, the normalization can still be of use. For example, suppose one feature is measured in centimeters, and another in meters. The one measured in centimeters will likely show much more variation. This is a trivial example and in a case like this one should always use only centimeters or only meters. But when features describe qualitatively different properties, it is not always obvious how to scale them. The variation about the mean is a unitless quantity, and it can be used to describe very different features in a uniform way.

Consider mean centering document vectors with binary features. This amounts to translating the unit hypercube. For each feature (word), we can project all points of the hypercube on the origin or on the point one. This gives the origin and the point one a mass. The sum of these masses is the mass of the corpus, the total number of documents. Now the hypercube is placed on the origin such that it doesn't topple. We can ask: which features vary most around their mean? Intuitively, these must be words that occur in half of the corpus.

2.9.3 Whitening

Whitening is closely related to the normalization described above. But here, one first locates the directions in space that capture most of the variance. These are called principal components. Latent semantic analysis, the subject of the next chapter, can be used to do this. Usually, one first mean centers the data, then one finds the principal components. The principle components are all orthogonal to each other, and form an alternative set of axes for the space. The document vectors may now be scaled along these axes, by dividing with the standard deviation along these axes. If the data were originally from a Gaussian distribution, then this process will yield a cloud of points shaped like a hypersphere. In the chapter on latent semantic analysis, we will explain the concepts introduced here in great detail.

2.9.4 Normalizing feature vectors to unit length

The conclusion and recommendation of a highly theoretical paper by (Herbrich & Graepel, 2002) is simple enough: "When training an SVM, always normalize your data in feature space". Feature space is the space in which the dot products are calculated with the kernel trick. Because we use only a linear kernel in our research, the feature space is just the input space. The kind of normalizing Herbrich & Graepel (2002) refer to is normalizing each feature vector to unit length. In the unigram word model with term frequency features, this would throw away the length of the document. It is a kind of transformation that can lose some data. For binary feature vectors, the transformation would not throw away information. After scaling a feature vector to unit length, the nonzero entries correspond to the words that were present and the original feature vectors could easily be obtained from unit length vectors again. It would be interesting to see if the normalization could nevertheless affect performance.

Chapter 3

Latent semantic analysis

In the previous chapter we discussed the vector space model. A limitation of this model is that if two documents contain different words with similar meanings, their feature vectors will be very different, even though the documents are conceptually related. Latent semantic analysis aims to capture similarity in meaning. It assumes that words that appear in a similar context will often be related. A very extensive treatment by one of its inventers from the perspective of human language learning can be found in (Landauer & Dumais, 1997). LSA was introduced in (Deerwester et al., 1990), mainly as a tool to improve information retrieval. In the field of information retrieval the technique is also known as latent semantic indexing.

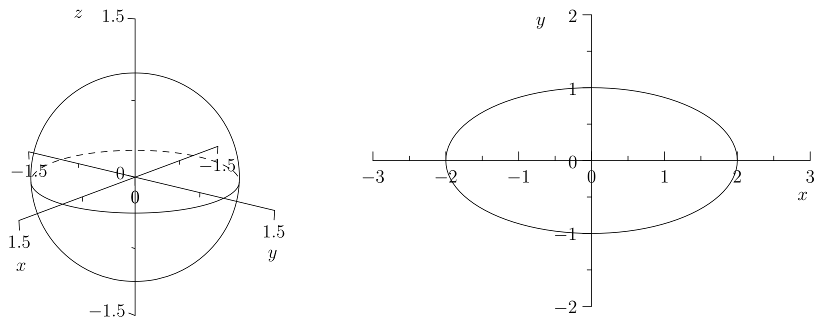
In this chapter, we will explain LSA in detail, but only a minimal background in linear algebra is required to understand the text. Mathematically, LSA is an application of a central result in linear algebra, the singular value decomposition (SVD) of a matrix. We will see how it can be used to project document vectors on a 'semantic space' (Landauer & Dumais, 1997) and how in this space the similarity between documents may be computed. It can also be used to project term vectors on a semantic space, and to calculate similarities between terms. The semantic space is a space of a lower dimension than both the document and the term vectors. Therefore LSA is a dimensionality reduction technique. We note that it is almost equivalent to principal component analysis (PCA), another dimensionality reduction technique.

3.1 The singular value decomposition of a matrix

An old result in linear algebra is that any matrix $M \in \mathbb{R}_{m\times n}$ with values from $\mathbb{R}$ can be factorized as

$$M = USV^T, \tag{3.1}$$

where V^T is the transpose of V, $V \in \mathbb{R}_{n\times n}$, $S \in \mathbb{R}_{m\times n}$ and $U \in \mathbb{R}_{m\times m}$. This equation can be a bit daunting if you are not too familiar with linear algebra. We will give an intuitive and visual explanation here that should be understandable with a minimal background in linear algebra. It is inspired by a great tutorial on the Internet by Todd Will (`http://www.uwlax.edu/faculty/`



(a) The unit sphere (b) A stretched ellipsoid

Figure 3.1: When vectors on the unit sphere in 3.1a are hit (left multiplied) by the matrix $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ they are projected on the ellipsoid in 3.1b. A matrix with only non zero values on its diagonal (in a non square matrix M the diagonal contains the values M_{ii}) is called a stretcher.

`will/svd/index.html`). We will work out the properties of the matrices U, S, and V step by step.

Any matrix $M \in \mathbb{R}_{m \times n}$ can be interpreted by the effect it has on a vector $\vec{v} \in \mathbb{R}^n$ in the multiplication

$$\vec{u} = M\vec{v}.$$

$\vec{u}$ is an m-dimensional vector, by the definition of matrix multiplication. In this way, a matrix is a function that transforms vectors from n- to -m dimensional space. In fact, it is a special kind of function, a linear transformation. **linear transformation**

Geometrically, a linear transformation can rotate and stretch a vector. Some matrices, however, only stretch vectors. These matrices are diagonal. We call them stretchers. See Figure 3.1 for an example. **stretcher**

orthogonal matrices Orthogonal matrices only rotate vectors, but do not stretch them. They are square matrices, and their rows (or columns) are orthogonal unit vectors. That is, they have length one and they are all perpendicular to each other.

If an orthogonal matrix has orthogonal unit vectors as its rows, we call it an aligner. **aligner** It would rotate the unit vectors of which it consists onto the standard axes of the space, and all other vectors would be rotated in the same direction. See Figure 3.2 for an example. If the matrix has orthogonal unit vectors as its columns, it is a hanger. **hanger** Such a matrix rotates the standard axes of the spaces onto its orthogonal unit vectors. See Figure 3.3 for an example.

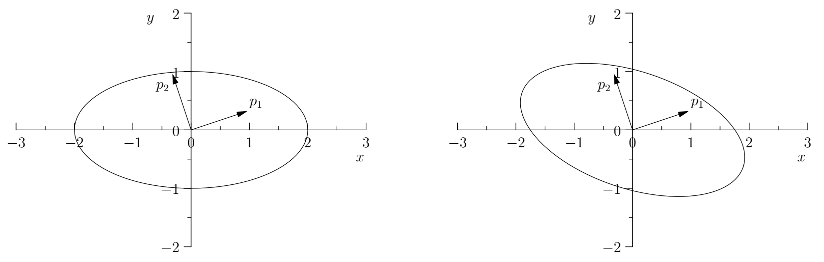
Now consider the singular value decomposition of $M \in \mathbb{R}_{m \times n}$ again:

$$M = USV^T.$$

The right hand side of the equation is a matrix multiplication. How to understand what a product of matrices does to vectors? The answer: composition: **composition**

$$(USV^T)\vec{v} = U(S(V^T\vec{v})).$$

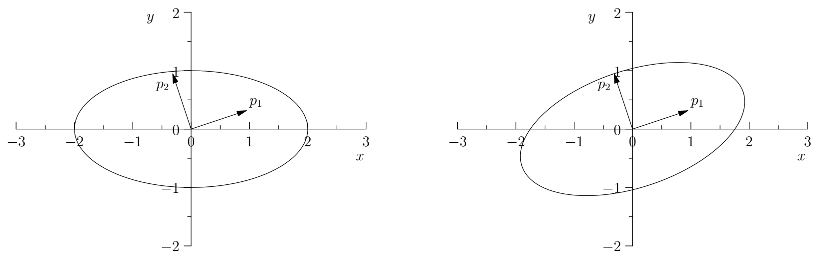
USV^T transforms a vector in three stages: first $V^T \in \mathbb{R}_{n \times n}$ transforms it, then $S \in \mathbb{R}_{m \times n}$, then $U \in \mathbb{R}_{m \times m}$. What are the properties of U, S and V? It turns out that V^T **is an aligner,** S **is a stretcher, and** U **is a hanger.**



(a) An ellipsoid and an alternative coordinate system (a perphrame)

(b) The same ellipsoid rotated towards the standard axes.

Figure 3.2: Example of an aligner matrix. The orthogonal unit vectors $p_1 = \begin{pmatrix} 3/\sqrt{10} \\ 1/\sqrt{10} \end{pmatrix}$ and $p_2 = \begin{pmatrix} -1/\sqrt{10} \\ 3/\sqrt{10} \end{pmatrix}$ form an alternative coordinate system for the space, a perphrame. An aligner matrix has p_1 and p_2 in its rows. When a vector is hit by an aligner matrix, it will be aligned with the x- and y-axis. If a vector was expressed in the standard coordinate system, multiplying it with the aligner matrix will give you the coordinates in terms of p_1 and p_2



(a) An ellipsoid and an alternative coordinate system (a perphrame)

(b) The same ellipsoid 'hung' on the perphrame

Figure 3.3: Example of a hanger matrix. The orthogonal unit vectors $p_1 = \begin{pmatrix} 3/\sqrt{10} \\ 1/\sqrt{10} \end{pmatrix}$ and $p_2 = \begin{pmatrix} -1/\sqrt{10} \\ 3/\sqrt{10} \end{pmatrix}$ form an alternative coordinate system for the space, a perphrame. A hanger matrix has p_1 and p_2 in its columns. When a vector is hit by a hanger matrix, it will be 'hung' on the perphrame. If a vector was expressed in the alternative coordinate system, multiplying it with the hanger matrix will give you the x-y coordinates.

singular value

The nonzero values on the diagonal of S are called singular values. There are only $\min(m, n)$ entries on the diagonal of S, so a matrix has at least one but at most $\min(m, n)$ singular values. The set of singular values is completely determined by the matrix M up to their sign. It is common to order the singular values in S from high to low, which makes S unique.

If we right multiply both sides of the SVD with V, we get

$$MV = USV^TV = US.$$

$V^TV = I$ because V is an orthogonal matrix. These matrices only rotate vectors, and their transposes rotate the vectors in the opposite direction. Let $\sigma_j = S_{jj}$ be the j'th singular value. Now note that the j'th column of MV may be written as $M\vec{v_j}$, where $\vec{v_j}$ is the j'th column of V. The j'th column of US equals $\sigma_j\vec{u_j}$, where $\vec{u_j}$ is the j'th column of U. We have

$$M\vec{v_j} = \sigma_j\vec{u_j} \tag{3.2}$$

singular vector

for $1 \leq j \leq \min(m, n)$. The unit vectors $\vec{v_j}$ and $\vec{u_j}$ are the right and left singular vectors corresponding to σ_j, respectively.

The SVD of a matrix has several interesting applications:

rank

- Estimating the 'true' rank of the matrix M. The rank of a matrix corresponds to the number of linearly independent vectors in its rows (or columns). For example, in a $3x3$ matrix, if the three rows or the three columns lie in one and the same plane, the matrix has rank 2. In M, if (for example) due to measurement errors some rows of a matrix are actually almost linear combinations of other rows, this can be hard to spot. But in the singular value decomposition this will be immediately obvious, since some diagonal values will be very close to zero.

- Low rank approximation. This application of the SVD is central to latent semantic analysis, so we will treat it in some detail. If we have the SVD of a matrix, we can throw away some of the smallest singular values (set them to zero) and maintain only the largest k in S_k. Then $k < min(m, n)$ and
$$M \approx \hat{M} = US_kV^T$$
is a low rank approximation of M. Now, if we keep the first k columns of U in $U_k \in \mathbb{R}_{m\times k}$, we truncate S_k to dimension kxk and we keep the first k colums of V in $V_k \in \mathbb{R}_{n\times k}$ then that leaves the result of the matrix multiplication unchanged, as it is easy to verify. So we write:
$$M \approx \hat{M} = U_kS_kV_k^T.$$
Since S_k is a matrix of rank k, we have that $\hat{M}$ is of rank k, because one of the properties of the rank of a matrix is that the rank of the product of two matrices is less than or smaller to the rank of either matrix. A commonly used measure for the magnitude of a matrix is the Frobenius norm. The Frobenius norm for an m by n matrix M is given by:
$$\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}^2}. \tag{3.3}$$

It has been proven that the matrix $\hat{M}$ that we obtained above minimizes the Frobenius norm of the matrix $M - \hat{M}$ for any matrix $\hat{M}$ of rank k.

Manning & Schütze (2000) compare low rank approximation by singular value decomposition to least squares approximation. If a matrix is of rank k, then its rows (and its columns) lie in a k-dimensional space. The rows (and columns) of M are projected on a hyperplane such that they have the smallest possible Euclidean distance relative to their original position. One can intepret this as modeling M with $\hat{M} = U_k S_k V_k^T$ plus some Gaussian error.

- Calculating the pseudoinverse. The inverse of a matrix A is denoted A^{-1} and it exists only if A is a square matrix of full rank. If it exists, then $AA^{-1} = I$. The pseudoinverse exists for any matrix and it is based on the SVD. The key idea is that even though S is not square, if we define $S^{-1} \in \mathbb{R}_{n\times m}$ with $S_{ii}^{-1} = \frac{1}{S_{ii}}$ and all other entries zero, $SS^{-1} = I$ and $S^{-1}S = I$. Now, if we right multiply the SVD with $VS^{-1}U^T$ we get

$$MVS^{-1}U^T = USV^TVS^{-1}U^T = I$$

  This means we can now define the pseudoinverse of M as $VS^{-1}U^T$. The pseudoinverse itself has many applications.

A more complete overview of the applications of the SVD can be found in (Leach, 1995).

3.2 The term by document matrix and the tf-idf measure

Now we will make things a little more concrete. Suppose M is a term by document matrix. There are m terms, and n documents. Its rows are term vectors t_i, its columns are document vectors d_j. M_{ij} indicates how relevant term i is for document j. The simplest measure for this relevance is the frequency of occurrence of term i in document j, but this has certain disadvantages. For instance, words such as 'the', 'a', 'for', etc. will be relevant to any document. Better measures also take into account global properties of the terms, such as the frequency of their overall occurrence. We will use the ubiquitous tf-idf measure, the term frequency-inverse document frequency measure. The term-frequency part is simply the frequency of term i in document j normalized by the total number of words in document j: **tf-idf**

$$tf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}}$$

where n_{ij} is the frequency of term i in document j. To scale down the importance of words that are very common in the entire corpus, a global measure of term i is used, the inverse document frequency part:

$$idf_i = \log \frac{|D|}{df_i}$$

where D is the set of documents, and $df_i = |\{d : t_i \in d\}|$ is the number of documents that contain term i. This logarithm decreases as the number of documents in which term i occurs increases. If there is no document in which t_i occurs this could lead to a division by zero. In our case, this cannot occur, because we base our vocabulary on the corpus itself. A common solution to this problem is to use the formula $1 + df_i$ in the denominator. The logarithm scales the idf_i measure down for low df_i, but it also makes idf_i decrease slower as df_i increases.

Finally,

$$\text{tf-idf}_{ij} = (tf_{ij})(idf_i)$$

so that tf-idf frequencies of words that occur much throughout the corpus are downweighted. We define our term document matrix M by

$$M_{ij} = \text{tf-idf}_{ij}$$

Now consider the matrix $MM^T \in \mathbb{R}_{m \times m}$. It is the term by document matrix multiplied by its transpose. By the definition of matrix multiplication $(MM^T)_{ij} = \vec{t_i} \cdot \vec{t_j}$, which is a measure of correlation between term i and j. This makes the matrix interesting. What does it do as a linear transformation? It takes a document vector $\vec{d} \in \mathbb{R}^m$ and gives another document vector $\vec{d'}$. Since

$$\vec{d'_i} = \sum_{k=1}^{m} (MM^T)_{ik} \vec{d_k} = \sum_{k=1}^{m} (\vec{t_i} \cdot \vec{t_k}) \vec{d_k}$$

we see that $\vec{d'_i}$ is boosted if $\vec{d}$ has high term weights for terms that correlate with term i. In a way, this results in a better document vector. Even if a term i is not present in $\vec{d}$, if similar terms are present, $\vec{d'_i}$ will be increased. In an information retrieval scenario, where we could have a query represented as a document vector, $\vec{d'}$ will be relevant to more terms than $\vec{d}$.

So far we have seen stretchers, matrices that only stretch any input vector, and orthogonal matrices, matrices that only rotate any input vector. These are very special matrices. Most matrices will stretch and rotate their input vectors. However, for many square matrices there are at least some vectors that are not rotated, only stretched. These vectors are called eigenvectors. Since an eigenvector $\vec{v}$ of a matrix A is only stretched by A, $A\vec{v} = \lambda\vec{v}$ for a certain λ. λ is the eigenvalue corresponding to $\vec{v}$.

eigenvector

eigenvalue

MM^T is a square matrix, and it is symmetric, because the dot product is commutative: $\vec{t_i} \cdot \vec{t_j} = \vec{t_j} \cdot \vec{t_i}$. This entails that it is a normal matrix. A matrix A is normal if it 'commutes with its tranpose': $A^T A = AA^T$. Since a symmetric matrix equals its transpose a symmetric matrix is also a normal matrix.

normal matrix

A central theorem in linear algebra, the spectral theorem, states that a normal matrix A can be factorized as follows:

spectral theorem

$$A = B\Lambda B^T$$

where B is an orthogonal matrix whose columns are eigenvectors of A, and Λ is a diagonal matrix that contains the corresponding eigenvalues.

For MM^T, a normal matrix, this means there are m normalized document vectors that are only scaled, not rotated by MM^T. They contain exactly such a mix of term weights that none of them are boosted relative to each other. In LSA, these eigenvectors are interpreted as concepts. They are a linear combination of terms. **concepts**

3.3 LSA: the SVD of the term by document matrix

There is a link between the SVD of M and the matrix MM^T: the eigenvectors of MM^T are the columns of U. This can be seen directly from the SVD and the spectral theorem:

$$MM^T = (USV^T)(USV^T)^T = USV^TV^{T^T}S^TU^T = USS^TU^T$$

Here we used the fact from linear algebra that $(ABC)^T = C^TB^TA^T$ and the fact that $A^TA = I$ if A is an orthogonal matrix. Now we know that SS^T is diagonal, on the diagonals it contains the squares of the singular values of M, as is immediately clear from the definition of matrix multiplication. By the spectral theorem, SS^T must contain the eigenvalues of MM^T on its diagonal. Thus, the singular values of a matrix M are the square roots of the eigenvalues of MM^T.

We also know that U is an orthogonal matrix. Then, by the spectral theorem U must contain as its columns eigenvectors of MM^T. Note that if MM^T does not have full rank, the set of eigenvectors is not completely determined. Thus, in the SVD of M, U and V are not always unique.

We could study with a similar approach the matrix $M^TM \in \mathbb{R}_{n\times n}$. We would find that it contains the correlations between document vectors. Two documents correlate well if they tend to contain the same terms. M^TM would transform a term vector to a 'better' one boosting term-document weights of documents that correlate well with the documents in the term vector. The eigenvectors of the matrix M^TM are the columns of V in the SVD of M and the singular values of M equal the square roots of the eigenvalues of M^TM. These eigenvectors are term vectors, they are linear combination of documents. They, too, are interpreted as concepts in LSA.

Now we can describe what $M = USV^T$ does to a term vector $\vec{v}$ geometrically: first, it is hit by the aligner V^T. This expresses the vector as a point in the term concept space spanned by the eigenvectors of M^TM. Then, it is mapped onto the m-dimensional space by S, and each of the m dimensions is scaled by a singular value. The vector is now in the document concept space of the eigenvectors of MM^T, where each dimension is a linear combination of terms. It is now hit by the hanger U, which expresses it again as a document vector, where each dimension is a term.

We have seen that it is possible to achieve a good low rank approximation of M by discarding the smallest singular values. These singular values correspond with eigenvectors in U and V which are then also discarded. We have, as before

$$M \approx \hat{M} = U_kS_kV_k^T.$$

If we let $k = \min(m, n)$ then $\hat{M} = M$. As we drop k the approximation becomes less precise. But not necessarily less useful. Which value to choose for k is a matter of experience, it depends on the application. In literature, values of anywhere between 200 and 1500 have been used succesfully. See Bradford (2008) for an overview of optimal values for k found in various research. In our research, we will also experiment with different values for k. Intuitively, k represents the number of concepts that are required to model the corpus in enough detail.

Observe that the j'th column of $\hat{M}$, which is an approximated document vector $\hat{\vec{d}}$, corresponds with the j'th column of V_k^T, a k-dimensional vector which we will denote with $\vec{d_k}$:

$$\hat{\vec{d}} = U_k S_k \vec{d_k}.$$

We see that $\vec{d_k}$ is stretched by S_k and then hit by the hanger-like U_k (U_k need not be square anymore, since some columns may have been dropped). In other words, it was expressed in terms of the columns of U_k, in the document concept space, but it is translated back to the standard coordinate system.

In LSA, we can use U_k to project document vectors like $\hat{\vec{d}}$ onto a k-dimensional concept space. If we left multiply both sides of the above equation with $S_k^{-1}U_k^T$, we have:

$$\vec{d_k} = S_k^{-1} U_k^T \hat{\vec{d}} \tag{3.4}$$

Note that U_k^T acts as an aligner here. Similarly, we can project our term vectors $\hat{\vec{t}}$ using V_k:

$$\vec{t_k} = S_k^{-1} V_k^T \hat{\vec{t}} \tag{3.5}$$

Deerwester et al. (1990) used equations 3.4 and 3.5 to produce their famous image where terms and documents are plotted together in a two dimensional concept space. But it is important to understand that it is not actually the projected term and document vectors $\vec{t_k}$ and $\vec{d_k}$ that we are interested in. We are interested in the approximation of the original term by document matrix M. In the next section we will see that the document-document similarities are computed using the columns of $\hat{M}$ and term-term similarity using the rows of $\hat{M}$. Finally, the $\hat{M}_{ij}$ is interpreted as the term-document similarity of term i and document j. The k-dimensional projected vectors are just an aid for easier computation.

3.4 Calculating similarities in semantic space

Deerwester et al. (1990) discuss how to compute similarities using the projected term and document vectors. The important thing to note is that this is based on the rows and columns of $\hat{M}$. It is then easy to show that the projected term and document vectors may serve as an aid for computation. This will be a lot cheaper in computation time, since k is usually much smaller than both the number of documents and the number of terms.

For document-document similarity, consider the dot product of two columns of $\hat{M}$, $\hat{\vec{d_i}}$ and $\hat{\vec{d_j}}$. By definition, it holds that

$$(\hat{M}^T\hat{M})_{ij} = \hat{\vec{d_i}} \cdot \hat{\vec{d_j}}.$$

Since

$$\hat{M}^T\hat{M} = (USV^T)^T USV^T = VSU^T USV^T = VS^2V^T = (VS)(VS)^T,$$

we have that

$$\hat{\vec{d_i}} \cdot \hat{\vec{d_j}} = ((VS)(VS)^T)_{ij}.$$

This is the dot product between row i and j of the matrix VS. A row of this matrix contains a projected document vector like $\vec{d_k}$, in the notation we used earlier, but stretched by S. Deerwester et al. (1990) say that the stretched projected document vector is 'in VS space', but this is a bit misleading. We saw earlier that

$$\vec{d_k} = S^{-1}U^T\hat{\vec{d}},$$

So for the stretched projected vector $S\vec{d_k}$ we have:

$$S\vec{d_k} = U^T\hat{\vec{d}}.$$

We know that U^T is an aligner, so $S\vec{d_k}$ is actually the full dimensional document vector projected on the space spanned by the left singular vectors! This can be confusing, because $\vec{d_k}$ is a row of V, and the columns of V contain the right singular values.

For term-term similarity, the rows of $\hat{M}$ are used as data points. The dot product between two rows is an entry of the matrix

$$\hat{M}\hat{M}^T = USV^T(USV^T)^T = USV^TVSU^T = (US)(US)^T$$

and equals the dot product between the corresponding rows of the matrix US. A row of the matrix US contains the projected term vector $\vec{t_k}$, but stretched by S: $S\vec{t_k}$. Since

$$\vec{t_k} = S^{-1}V^T\hat{\vec{t_i}}$$

we have that

$$S\vec{t_k} = V^T\hat{\vec{t_i}}.$$

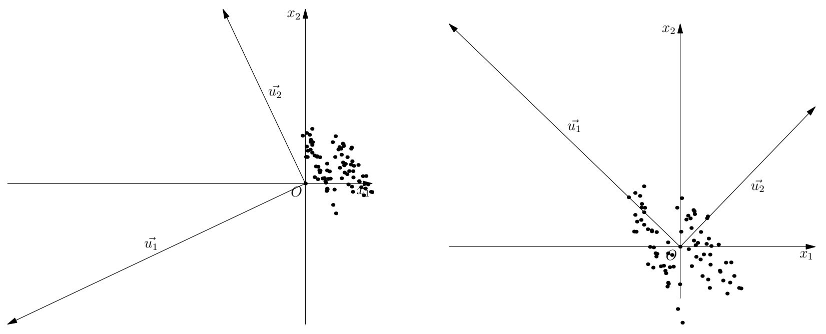
Because V^T is an aligner, the rows of US are the full dimensional term vectors projected on the space spanned by the right singular vectors.

It is also claimed that LSA can compute term-document similarity. The quantity of interest for this similarity is nothing but: the term-document weight in the approximated matrix $\hat{M}$. While this quantity may not be so impressive in the original matrix M, in the lower dimensional space one hopes that the quantity is enhanced if the term occurs often with other terms that also occur in the document. Once again, the lower dimensional vectors may serve as a computation aid, via a nifty trick. Since

$$\hat{M} = USV^T = US^{\frac{1}{2}}S^{\frac{1}{2}}V^T = US^{\frac{1}{2}}(VS^{\frac{1}{2}})^T,$$

we have that $\hat{M}_{ij}$ is the dot product between row i of the matrix $US^{\frac{1}{2}}$ and row j of the matrix $VS^{\frac{1}{2}}$. For computing term document similarities it is then necessary to use all three the matrices U, S, and V. The derivations of the computation of the three different types of similarity above was taken from Deerwester et al. (1990). It was repeated here to show an extra step here and there, but also to make clear that the similarities are not all calculated in the same space. Also, we stressed that document similarities are calculated in the space spanned by the left singular vectors, and term similarities in the space spanned by the right singular vectors. In the next section we will see that each of these spaces can also be found by doing a principal component analysis (PCA) on the document vectors or the term vectors, respectively. The analogy with PCA will point naturally to a preprocessing step that would seem advantageous for LSA as well: to center the data points around the origin first.

3.5 LSA, principal component analysis and mean centering



(a) The left singular vectors of the SVD

(b) The left singular vectors of the mean centered data.

Figure 3.4: If the points plotted are loaded in the columns of a matrix M, then $\vec{u_1}$ and $\vec{u_2}$ are the left singular vectors of the SVD of M. In 3.4a one can see that u_1 intersects with the mean of the points. In 3.4b the data has been mean centered before calculating the SVD. The direction of the left singular vectors found now coincide with the principal axes found by PCA.

In the previous section, we saw that when we compute document-document similarities, we consider the columns of $\hat{M}$ as data points, and we work with the left singular vectors in U. In Figure 3.4a we see a cloud of two dimensional datapoints, and the left singular vectors that an SVD on the corresponding matrix finds. The vectors are scaled by their corresponding singular values. The singular values give an indication of how much variance in the data is 'explained' by the singular vector. However, u_2, the smaller of the two singular vectors plotted seems closer to the main direction of variance in the data points than u_1. This is because the data was not mean centered before calculating the SVD. In the LSA literature (Deerwester et al., 1990), (Landauer & Dumais, 1997) there is often no mentioning of mean centering the data before doing the analysis.

In principal component analysis, however, mean centering the data is very common. Miranda et al. (2008) discuss mean centering in research praxis with PCA, noting that it is often done as a preprocessing step. In one popular approach to finding the principal components it is implied. Here, one starts with the sample covariance matrix of the data points. The covariance of features is the degree to which features of the document vectors covary around their mean. If we have a term by document matrix $M \in \mathbb{R}^{m \times n}$ and we want to do a PCA on its columns, the document vectors d_j, then the sample covariance matrix $\sum$

sample covariance matrix

would be defined by

$$\sum = \frac{1}{n-1}\sum_{i=1}^{n}(\vec{d_i} - \overline{\vec{d_i}})(\vec{d_i} - \overline{\vec{d_i}})^T,$$

where $\overline{\vec{d_j}}$ is the mean document vector. It turns out that if we mean center our documents before doing an SVD, then the columns of M are vectors $\vec{d_j} - \overline{\vec{d}}$ and then

$$\sum = \frac{1}{n-1}MM^T.$$

The next step in PCA is to compute the eigenvectors of the sample covariance matrix. These are the principal components. But we have seen before that in the SVD of M, U contains the eigenvectors of MM^T in its columns! Therefore, the left singular vectors of the SVD point in the same direction as the principal components. In fact, the SVD of mean centered data is widely employed to calculate the principal components (Miranda et al., 2008). An SVD gives the solution to two PCA problems at once: one in which the documents are seen as data points and one in which the terms are seen as data points. In LSA, if we compute term similarities, then we use the right singular vectors, and these are the principal components of the term vector points.

Miranda et al. (2008) also note that it is necessary to mean center the data for an optimal low rank approximation. In Figure 3.4b the data is mean centered before calculating the LSA. Here we can indeed visually confirm that $\vec{u_1}$ is the line that gives the projection of the data points with the smallest squared error in Euclidean distance. It also shows clearly that LSA is a form of least squares, specifically total least squares approximation. It could be that in information retrieval mean centering of the data is not commonly done, and doing it could prove to be advantageous. However, even if the first singular component is not very informative with data that was not mean centered, the remaining principal axes still have many degrees of freedom $(m - 1)$ in the high dimensional space, so the problem may not be big. Mean centering the data does have a practical disadvantage. While a typical term by document matrix is very sparse, after mean centering it will be a full matrix. A full matrix consumes much more disk space.

total least squares

Now that we have a clear understanding of latent semantic analysis, in the next section we will take a look at how we may use it in a supervised learner such as a support vector machine.

3.6 Using LSA with support vector machines

Looking at Figure 3.4b, we see that the cloud of points seems to consist of two separate clouds. In fact they were generated by two separate Gaussians. Suppose that the left points are from one class, and the right points from another. The points are linearly separable, and an SVM could be succesfully trained on them. But if we approximate the data with LSA with an SVD of rank 1, there is no way a hyperplane could separate the points anymore. If we are not careful, then, a principal component analysis could be harmful. LSA and PCA are unsupervised techniques, they do not take the class labels of training instances into account. They are used for a different purpose than an SVM. Whereas an SVM tries only to classify the data, LSA and PCA try to model the data, summarize the data.

LSA and PCA keep only the directions in the original space that contain the most of the variance of the data. Because they reduce the dimensionality of the problem, they can be beneficial as a preprocessing step for many machine learning algorithms. But of all possible algorithms for classifying reviews, we chose support vector machines for their excellent performance. Support vector machines are not very sensitive for high dimensionality because they control their VC-dimension by maximizing the margin of the hyperplane. Therefore we do not expect preprocessing with LSA to be a substantial improvement. It could even make performance worse, because, as in the example from the previous paragraph, the directions that explain most of the variance need not coincide with the directions that predict class membership best. This information could be contained in some of the directions that correspond with one of the smaller singular values that we discarded.

In movie reviews, we expect topic-sentiment interaction to occur. For comedy movies, it would be good if the scenes are funny, and the characters charming. For thrillers it would be good if the scenes are terrifying, and the characters creepy. The idea of LSA is that it models underlying topics in a corpus. These are linear combinations of words that explain most of the variance in the corpus. We expect the topics of the movies that are being reviewed to determine the words in the reviews to a large extent. This motivates the idea of adding, say, the top 10, or the top 100 top principal components to the feature vector of a document. These principal components could be calculated for the entire corpus of 27000+ reviews. The reviews that have to be classified may be projected on these axes. This might help a support vector machine to classify the reviews better. There could be a problem with this approach, too, however.

The problem is that whatever the direction of the singular vectors may be, they remain linear combinations of the original features. Consider a one dimensional data set. A bunch of '+' and '-' datapoints on a line. If we add another dimension to this dataset that is the original feature multiplied by a constant, then the datapoints will be lifted, but they will still lie on a line. In general, if we add dimensions that are linear combinations of the original dimensions, the dimension of the subspace that the points were in will not increase. Similarly, the rank of a matrix does not increase if you add rows or columns that are linear combinations of other rows or columns, respectively. Therefore, a set of points that was linearly inseparable in the original space will probably remain linearly inseparable in the augmented space. And a set of points that is linearly separable in the augmented space is probably also linearly separable in the original space.

Is using LSA in combination with SVMs simply a bad idea then? Not necessarily. The document similarities that can be computed with LSA should improve as the corpus size on which the SVD was computed increases. The idea is that in the lower dimensional space documents that share no concrete terms can nonetheless be close together if they map on the same 'concepts'. This can

prototypes

be exploited by working with prototypes. Prototype based learning characterizes test points by their distance to known prototypes. K-nearest neighbour is an example of this approach. Another example are support vector machines that use a radial basis function as a kernel. The hope is that the distance measure through LSA is superior to a distance measure in the original vector space. In the next chapter we treat semantic orientation and we will see that it amounts to putting some prototypes in LSA space and calculate the distance towards them. By adding these features to the original document vector performance may be increased a little. But before we treat semantic orientation, we will make some closing remarks on LSA in natural language processing.

3.7 LSA and natural language

synonymy

In information retrieval, synonymy sometimes poses a problem. For example, in a document about LSA, the term LSI, which is a synonym for LSA, might not be present. Still, if someone would search for LSA, documents about LSI are relevant. In the vector space model, where each term is a dimension, the query and the document might be further apart than desirable. LSA alleviates this problem.

To grasp the intuition behind this, remember what the matrix MM^T, which contained the correlations between term vectors, did to a document vector $\vec{d}$. It boosts term weights that are omitted, but that correlate well with terms that are present in the document vector. If one of a number of synonyms is omitted, one would expect it to be boosted by MM^T. In other words, the document vector would be rotated. If weights are distributed more equally among synonyms, a vector would be rotated less. The eigenvectors, remember, are the vectors that are not rotated at all. These are the axes of the concept space. One hopes, therefore, that documents that contain only some terms of a set of synonyms, will be projected close to each other in the concept space. LSA was designed with the problem of synonymy in mind, it was intended to optimize recall in a document retrieval setting (Deerwester et al., 1990).

first order cooccurrence

second order cooccurrence

In fact, LSA aims higher than just solving synonymy. It aims to project terms that have a similar meaning close to each other in concept space. If terms occur together in documents often (first order cooccurrence), their term vectors will correlate well. In case of synonyms, we have seen that their first order cooccurrence need not be high. If there is a set of words, however, that cooccur much with each other, and each of them with the synonyms, than their second order cooccurrence will be high. The SVD of a matrix captures this, as well as higher order cooccurrence.

Denhière & Lemaire (2006) ask how important higher order cooccurrence is for determining term similarity. They define word similarity in the context of language learning as associative strength between words in the human brain. A high similarity of two words would then imply that if one of the words is presented, the other would have a high activation in memory. In research, term similarity is often approximated with measures that use only first order cooccurrence. In an experiment Denhière & Lemaire (2006) tried to measure how much higher order cooccurrence influenced term similarity. The conclusion was that although first order occurrence explains most of it, higher order cooccurrence contributes significantly.

polysemy

Another problem is polysemy. One word can have very different meanings. In a document retrieval setting, if a query consists of just one such word, there is no way any algorithm could know what meaning the user intended. An ideal algorithm would, however, detect that the word is polysemous, and would group search results by meaning. LSA offers no such solution, in term concept space, each original term vector is still only represented by one vector. If a query consists of multiple words, one or more of which are polysemous, LSA does offer some advantages over the vector space model. While multiple dimensions in the concept space may have the polysemous term(s) in their linear combination, due to randomness, one may hope that dimensions that do not correspond to the intended meaning somehow 'cancel out'.

Experience with LSA suggests that paragraphs are suitable text units for the document vectors, because in paragraphs text is often semantically related. For us, it will be more convenient to use whole reviews as documents, because the reviews have labels, and we intend to use LSA in a semi supervised learning framework. This is also common practice, and LSA is robust also for longer texts (Landauer & Dumais, 2008).

3.8 Semantic Orientation (SO)

In the previous section, we have studied latent semantic analysis (LSA). We have seen that it can be used to calculate a similarity measure between two terms, two documents, or a term and a document. Now as an intermezzo we will discuss semantic orientation. In the last section of this chapter we will show how semantic orientation may be calculated with document similarities in the LSA document concept space.

Turney & Littman (2003) give the following formula for calculating what they call the semantic orientation from association SO_A of a word w towards two opposite sets of words:

$$\text{pwords} : \{good, nice, excellent, positive, fortunate, correct, superior\}$$
$$\text{nwords} : \{bad, nasty, poor, negative, unfortunate, wrong, inferior\}$$

$$SO_A(w) = \sum_{w_i \in pwords} A(w, w_i) - \sum_{w_i \in nwords} A(w, w_i), \qquad (3.6)$$

where $A(w, w_i)$ is a function that gives the strength of association of the words w and w_i. This association is the heart of this formula. Turney & Littman (2003) evaluate two measures of word association in an experiment, where words from the 'Positiv' and 'Negativ' categories of the lexicon of the General Inquirer had to be classified correctly (a positive SO_A score is interpreted as a prediction that the word is positive).

pointwise mutual information

One of the association measures tested was pointwise mutual information (PMI) (Church & Hanks, 1990). It counts the number of times the two words cooccur in a corpus, and then calculates a log odds ratio:

$$PMI(w_1, w_2) = \log \frac{P(w_1 \cap w_2)}{P(w_1)P(w_2)}$$

Turney & Littman (2003) queried the AltaVista search engine using the NEAR operator with two words. The NEAR operator only returns snippets of text in which both words occur in a radius of ten words. The cooccurrence count is then simply the number of hits returned. Because the index of Altavista is huge, this leads to reliable probability estimates. Unfortunately, AltaVista has since discontinued its NEAR operator (Taboada et al., 2006).

The other association measure that Turney & Littman (2003) tested was the similarity of two terms in an LSA space that was constructed from a corpus of around 10 million words in short documents of various kinds. They used around 250 dimensions. We know already that an advantage of LSA is that it also takes into account higher order cooccurrence. Although PMI worked better on the Altavista corpus, when PMI and LSA were both computed with the smaller 10 million word corpus, LSA outperformed PMI.

Classifying words as positive or negative is one thing, classifying reviews as positive or negative is another. Turney et al. (2002) extracted word pairs consisting of an adjective or an adverb coupled with a noun from reviews. This was done because in previous research adjectives had proven to be good indicators of subjectivity in a sentence. A noun was required to give some context. Because Turney et al. (2002) evaluated reviews for entirely different products, context was useful. A good example they give is the adjective 'unpredictable', which is positive in the phrase "unpredictable plot" in a book review, but negative in the phrase "unpredictable steering" in a car review. With AltaVista's NEAR operator, calculating association between a word and a phrase is possible, because the phrase can be enclosed by double quotes.

The semantic orientation of the review was simply the average semantic orientation of the two word phrases, where *pwords* was just the word 'excellent', and *nwords* the word 'poor'. If it was positive, the review was classified as positive. This is an unsupervised algorithm, but Turney et al. (2002) note that SO_A may be used as a feature in a supervised algorithm. In the latter case, it would seem that it can be interesting to use the average association of phrases in the review with *pwords* and the average association with *nwords* as two separate features. For example, if both associations are high, the reviewer might be more excited than if they are both low, even though the SO_A score would be the same in both cases. Especially when SO with regard to emotionally charged terms is calculated, this approach may give more information.

With LSA, it is much harder to incorporate bigrams. Also, using bigram frequencies would require a much larger corpus than the corpus of around 23000 reviews we use in this research. Still, the semantic orientation of words in a review should be useful, intuitively. In the next section we will see that in the LSA concept space, semantic orientation of a review can be elegantly computed.

3.9 Calculating SO in the LSA concept space

Consider again equation 3.6. If *pwords* and *nwords* would have different size, the equation would not be 'fair', the larger set would have more contributing factors. If we want to allow a different number of words in *pwords* and *nwords*, we can take the average associations between a word w and the two sets. Let N_p be the number of words in *pwords*. Then the average association $\overline{A}$ of w with *pwords* is

$$\overline{A}(pwords, w) = \frac{1}{N_p} \sum_{w_j \in pwords} A(w_i, w_j)$$

The most straightforward way to calculate the association of a review with *pwords* is to take the average $\overline{A}$ for all N words in the review:

$$\begin{aligned}\overline{A}(pwords, review) = \frac{1}{N}\sum_{w_j \in review}\frac{1}{N_p}\sum_{w_i \in pwords} A(w_j, w_i) = \\ \frac{1}{N_p}\sum_{w_i \in pwords}\frac{1}{N}\sum_{w_j \in review} A(w_j, w_i) = \\ \frac{1}{N_p}\sum_{w_i \in pwords}\overline{A}(review, w_i).\end{aligned}$$

But we can improve on $\overline{A}(review, w_i)$. Not every word in the review is equally important for that review. We would like to use the term document weight there to get a weighted average. In other words, we would like to use the document vector of the review. And instead of averaging over all words in *pwords*, we may think of *pwords* as a pseudo-document. This means we can express it as a document vector $\vec{d}$, and we may compute a k-dimensional projection of it as we would have done with an ordinary document vector:

$$\vec{d_k} = S^{-1}U^T\vec{d}$$

folding in

This process is called folding in. Then instead of calculating term similarities, we may simply compute LSA document similarity between *pwords* and the review. While Turney & Littman (2003) was interested in comparing different similarity measures for words, we are commited to using LSA for computing similarities. Therefore, we are able to use the more natural document similarity.

The pseudo document *pwords* looks just like a prototypical positive review. We may explicitly label the review as a positive review and use it as an additional training instance in a regular SVM. However, it may be more interesting to add the LSA similarity of each review to *pwords* as a feature for an SVM. We could in the same way create a pseudo review *nwords* containing negative words. But we could also create a pseudo document $funny$ containing funny words. Or a set of words *scary* containing scary words. In this way, we add some additional information to the feature vectors that are fed to the supervised algorithm. Intuitively, these features could be helpful for classification. And these features do increase the dimension of the feature vector. In principle, they could make a linearly inseparable problem linearly separable.

The full feature vector also allows for interaction between funny words and genre-related words, so no great performance increase is to be expected. But marking some points in space as 'interesting' does seem to be a meaningful way to give a machine learning a bit more knowledge about the problem domain.

Chapter 4

Modeling reviews as bags of words

In the previous chapters, we have discussed support vector machines and latent semantic analysis. These techniques work with real valued vectors, as we saw in some of the toy examples. In this chapter we will discuss how we describe documents, movie reviews, with a vector. We chose to model reviews with the bag of words model. This is also called a unigram language model. Joachims (2002) shows that it can be used with much effect in the task of text categorization. In the bag of words model we first construct a vocabulary. Each word in the vocabulary is a feature. Given a document, we count how often each word occurs in it. If we then put the document vectors in the columns of a matrix, we get a term by document matrix.

In the next section, we discuss some limitations of the bag of words model. Then, we will describe our dataset and some specific challenges that it presents. We will see how to find the words in the reviews through a process called 'tokenizing'. We try to identify 'noise'. These are words that we think are irrelevant for determining if a review is positive or negative. In the end, this leads to a term by document matrix. Once we have that, we can transform it in various ways as discussed in the previous chapters, and obtain feature vectors that we can feed to classifiers in our experiments.

4.1 Limitations of the bag of words model

The bag of words model does not take into account word order. The two sentences "not bad, excellent" and "not excellent, bad" have the same bag of words representation. Should we not at least detect negation? Pang et al. (2002) experimented with adding specific bigrams. If a negation word was found in a sentence, then for the remainder of the sentence for each word a second special word was added: the concatenation of `not_` and the word. They found the effects to be negligable.

An alternative solution to the problem of negation was proposed by Liu & Seneff (2009). They collected a large dataset of 137.569 reviews on 24.043 restaurants. Based on presence of high frequency adjectives and nouns a subset of sentences from this corpus was selected. Then the sentences were parsed using a hand crafted context free grammar. 78.6% of the sentences was parseable with this grammar. Using the full parse of these sentences, they are able to find which clause a negation applies to even in sentences like: “I wouldn’t say the movie was very good.” In this sentence, they would find that the movie is “not very good”. The next question is what to do with this knowledge.

For this, they propose a linear additive model. For each adjective, a score is calculated on the basis of the average user rating of reviews in which it appears. For example, the word ‘good’ gets a score of 4.5. For each adverb, it is calculated if they make the adjectives they appear with stronger or weaker based on the difference in average user rating in reviews where the adverb is used and reviews where it isn’t used. The word ‘very’ receives a score of 0.5. Negation is treated similarly. It receives a score of -3.0, because on average, it makes adjectives a lot weaker. The score for the phrase “not very good” is then calculated as $4.5 + 0.5 + -3.0 = 2.0$.

Parsing sentences grammatically is one of the main alternatives to the unigram language model. People understand grammatical structure of sentences. If we want to rate reviews with the same accuracy as people, perhaps there is no other way. However, parsing is still a very hard problem. Liu & Seneff (2009) could parse less than eighty percent of the subset of sentences that contained high frequency adjectives and nouns. Parsing a sentence means that it is accepted by their context free grammar. Building this grammar involves considerable human knowledge. Still, a syntactical parse of a sentence does not yet mean that the meaning of a sentence can be inferred.

It is hard to compare the unigram word model with the parsing approach of Liu & Seneff (2009) since their dataset is very different from ours. Also, their task is different, as their ultimate goal is to predict how a review rates different aspects of the experience of dining at a restaurant. Parsing sentences is out of the scope of this research. Still, this is one of the main directions in which improvements may be sought.

It is not very hard to find other limitations of the bag of words model. For example, consider the phrase “Excellent, outstanding, yeah right”. We will see in chapter 5 that sarcasm can be a problem. Even the approach we just discussed would have trouble with phrases like this. Another problem we encountered was that when a reviewer compares the reviewed movie to other movies, it is not possible to tell which words refer to which movie. Solving this problem would require an approach at the sentence level, where it would be hard to do without some form of parsing. A related problem is anaphor resolution. An anaphor is an expression referring to another expression. For example: “His previous movie was excellent, way better than this one”. In a sentence like this, there are many references: ‘His’, “His previous movie”, “this one”. Anaphor resolution aims to identify that ‘His’ refers to the director, and “this one” referst to the movie being reviewed. It is an area of active research, and an unsolved problem.

4.2 Description of the dataset used

Pang et al. (2002) randomly selected 2000 reviews from a total of 27886 html files, but such that half of the selected reviews was clearly positive, the other half negative. They processed these reviews, tokenized them, and used them in their research. They also provide all the 27886 html files. These were downloaded from the Internet Movie Data Base (IMDB ,`http://www.imdb.com/Reviews/`). We used the grand dataset of 27000+ html files for the latent semantic analysis. We used the two thousand labeled reviews for classification.

The fact that half of the labeled reviews is positive and the other half negative means that the set of labeled reviews cannot really be seen as a random sample from a larger population of movie reviews. In reality, there are probably more positive than negative reviews. If somebody is enthousiastic about a movie, there is a greater chance that he or she will take the time to write a review about it. Because we nowhere use the knowledge explicitly that half of the reviews is positive, the approach described in this research may be expected to yield similar results in real life, however. For example, the transductive SVM algorithm described in chapter 2 estimates the total number of positive reviews from the fraction of reviews in the training set that is positive.

We don't know exactly how the dataset was created by IMDB. A casual search for internet addresses (URLs) using a regular expression yielded over 900 distinct URLs of mostly movie review sites. Many reviews also seem to have been posted to news groups. It appears that IMDB gathered the data from various sources. An html header and footer have been attached, in which some meta data is presented, such as the author of the review, and the title of the movie being reviewed.

More than 1500 different people authored the 27000+ reviews. Just over 300 different authors wrote the two thousand labeled reviews. Why is this interesting? For several reasons. In the unigram language model, we hope that the positivity or negativity of a review determines to some extent which words occur in a review. In the task of text categorization, where the unigram language model has been tried and tested, it has been shown that the topic of a document influences which words appear in it. Authors of course also determine which words are in the review. Maybe there are differences between male and female authors. Education and age may play a role.

In this research, we are not interested in predicting if an author is male or female. Still, there are two things that could play a role. First, some authors may mainly write critical reviews, while others might only write reviews for films they are enthousiastic about. If this would be the case, we would expect author names to pop up as discrimating features when we analyse afterwards what a classifier has learned. Second, most authors use their own rating scales in the reviews. That means there are possibly over 300 different systems of rating a movie. Some people rate on a scale from one to four or five stars. Others rate with letters, anywhere between A and Z.

In the entire dataset, there are reviews on more than six thousand movies. The two thousand labeled reviews rate just over 1100 distinct movies. Of course, a movie like "The Godfather" would be more likely to receive a high rating. We planned to sample a training and a test set such that in the test set no movies are reviewed that are also reviewed in the training set. This would prevent a classifier from using the movie titles. We dropped this plan, however. In a real life setting, this would often not be necessary. For example, if we want to measure opinion on a political party, we could use old documents about the same political party, too.

4.3 From bytes to characters

Every file on a computer is eventually just a bunch of zeros and ones: bits. Eight bits form a byte. A byte can have 255 values. A standard was developed in the sixties to agree on a translation of bytes to characters. This is the ASCII standard, or the ASCII encoding. It defines characters for the byte values 0 to 127. For example, the decimal value 32 is the space character in this encoding. Of course, 127 characters quickly proved way to little to encode every special character in every language. That is why, many different extensions to the ASCII standard gained in use.

Since the reviews originated from many sources, we may expect them to be encoded differently. In the English language, there aren't many special characters. Still, in movie titles or actor names there can be words from foreign languages. At any rate, in our corpus, well over six thousand documents contained byte values in the range $128 - 255$. This indicates that an encoding other than the ASCII encoding was used. It is good practice to indicate in your digital documents how they are encoded. Unfortunately, on the wild wild web, this is not a common practice. Only eight of the reviews contained character encoding information.

The python module 'chardet' [1] was used to make an educated guess on the character encoding of the reviews:

```
    1 EUC-TW
    1 Not found
    1 TIS-620
    2 ISO-8859-7
    2 MacCyrillic
    3 IBM855
    3 windows-1251
    4 Big5
    9 windows-1255
   24 utf-8
   45 SHIFT_JIS
 6143 ISO-8859-2 (82% confidence)
21648 ascii (100% confidence)
```

Of the six thousand ISO-8859-2 reviews the library isn't too sure and indeed these reviews often contained bytes that made no sense in this encoding. After making a histogram of the bytes in the dataset, some problem bytes were identified, and we translated these with the following table:

\xa9	the copyright symbol	<st>copyright</st>
\xa0	a non breaking space	a space
\x85	NEL (next line)	a newline
\xe2\x80\98	LEFT SINGLE QUOTATION MARK	a single quote
\xe2\x80\99	RIGHT SINGLE QUOTATION MARK	a single quote
\xe2\x80\9c	LEFT DOUBLE QUOTATION MARK	a double quote
\xe2\x80\9d	RIGHT DOUBLE QUOTATION MARK	a double quote
\x91	Windows smart left quote	a single quote
\x92	Windows smart right quote	a single quote
\x93	Windows smart left double quote	a double quote
\x94	Windows smart right double quote	a double quote

The first column of this table shows the hexadecimal value of the byte. The second column lists the most probable meaning. The third column lists the ASCII character that the byte was translated to.

[1] http://chardet.feedparser.org/

4.4 Parsing html

In the previous section, we saw how we handled the encoding of the reviews. In this section we will parse the review as html. We assume that the reader is familiar with html, or Hypertext Markup Language [2]. All reviews are in html format. All reviews contained at least two `<hr>` tags that marked the end of the IMDB header and the start of the IMDB footer, respectively. In addition, most reviews contained headers and footers from many different review sites.

These headers and footers contain information that is not relevant for determining if a review is positive or negative. Parsing html is a good start to get rid of some of this noise. We selected a Python module called BeautifulSoup [3] to do the parsing for us, because it advertises to be a very noise tolerant html parser. Nevertheless, a couple of hundred review sites did not parse correctly. For example, if the two `<hr>` tags were not detected by the parser, this meant that halfway some html tag would be corrupted, and only a small part of the review would be parsed. We discarded these reviews. Of the labeled reviews, we lost 55 reviews, leaving a total of 1945 labeled reviews.

boilerplate removal

Removing headers and footers of other movie sites turned out to be a not so trivial problem. The task has been called boilerplate removal (Fletcher, 2004). The CleanEval competition (Baroni et al., 2008), a competition for cleaning web pages, is devoted to it. Most researchers joining the competition submitted supervised machine learning algorithms. It is clear that boilerplate removal is an interesting research subject in itself. In this research, however, the aim was to classify reviews, and noise removal should not be much more than a preprocessing step.

One approach is to try to exploit the html structure of pages to identify what is boilerplate and what is natural language or connected text. We built a Site Style Tree following Yi et al. (2003). The html structure of a page is a tree. The Site Style Tree merges the trees of a collection of internet pages. It starts with a root, the `<html>` element. Then, for each page, the sequence of children of the root html element is added as a new child to the Site Style Tree. If the sequence already exists, its count of occurrence is incremented. Then, for each html element in the sequence, the sequence of children of that element is added to the Site Style Tree. The idea behind it is that many web pages have a surrounding template. This will often be an identical sequence of html elements. Natural text will show more variation in its html structure and its text. Therefore, sequences in the Site Style Tree that have a high occurrence count are probably noise.

For our corpus of reviews, the Site Style Tree did not work, however, because of two problems. First, html structure was too flat. If natural text occurs at the same nesting level as headers and footers, they are part of the same sequence of children. Second, our reviews originate from too many internet sites. Each site has its own boilerplate structure. Together, these problems caused us to drop the idea. Instead of working with the html structure of the review pages to remove boilerplate, we turned to a data structure called a suffix array. More about that in section 4.6 below.

After parsing the html and removing the IMDB headers and footers, we saved the reviews in a simplified html format. We kept the 'bold' tag (`<b>`) and

[2] `http://www.w3.org/html`
[3] `http://www.crummy.com/software/BeautifulSoup/`

the 'italics' tag (`<i>`) because they signal emphasis. Emphasis signals emotion and may be relevant in sentiment analysis. We kept the paragraph tag (`<p>`) because the division of connected text in paragraphs may be a useful one. Other tags we kept are `<pre>`, `<a>`, `<br>` and `<hr>`). Here is a fragment of a sample review (review 6370.html) after parsing html:

```
<pre>                              RANSOM
                   A film review by Michael Redman
   Copyright 1996 Michael Redman </pre>  <pre> ***1/2 (out of ****)
</pre> <p> If he doesn=92t watch out, Mel Gibson is in danger of being
known as someone other than Mad Max. Of course there are still a few big
guns in this film, but there=92s some accomplished acting too. Although
Gibson continues to shed his rough and gritty anti-hero skin as his
career progresses, here he explores a new genre: the businessman as
action hero. </p>
(...)
 <p> But you=92ll ignore these problems as each minute of the movie
builds on the previous one. Forget about the large coke, you won=92t
want to leave for that five minute break an hour later. </p> <p> [This
appeared in the 11/14/96 "Bloomington Voice", Bloomington, Indiana.
Michael Redman can be reached at <a> mredman@bvoice.com </a> ] </p>
```

Three things catch the eye. First, there is an explicit rating, which we want to remove. In the next section we'll see that these kind of ratings are caught with a few simple regular expressions. Second, something went wrong with quotes inside words. This kind of noise is hard to avoid completely when gathering pages from the wild web. Third, there is a little boilerplate at the bottom, intermingled with information on who is the author. This kind of boilerplate typically contains n-grams of more than a few words that appear more than a few times in the corpus. This is what we can highlight with a suffix array.

4.5 From characters to words: tokenization

Tokenization is the process of identifying words in a sequence of characters. In addition to finding the words, we also want to discard certain patterns, notably explicit rating patterns. Rating patterns often contain the asterisks. The asterisk is also often used to emphasize words in newsgroup messages, and many of our reviews were posted on newsgroups. We wanted to preserve emphasis because it can be a signal of a strong sentiment. The asterisk is also often used in ASCII art, for example as a horizontal rule. Finally, it is used to mask characters in words that would otherwise be considered offensive. These are usually words that are also strongly associated with sentiment, and we want to keep them in our vocabulary. For flexibility, we chose to implement our own tokenization.

Manning et al. (2008) wrote a book on information retrieval that is available online [4] and that is commonly known as the "IR book". It contains an excellent section on tokenization, section §2.2.1, with practical advice on how to handle quotes and hyphens, for example. In the next couple of paragraphs, we describe how we tokenized our reviews.

We used a lexical scanner generator that is a derivative of the standard unix utility 'lex': flexc++ [5]. A lexical scanner generator is given a set of rules in an input file that is called a lexer file. Each rule is a regular expression, and each rule is associated with an action. A regular expression can match a token,

[4] `http://nlp.stanford.edu/IR-book/information-retrieval-book.html`
[5] `http://www.flexcpp.org`

e.g. a word. An action can be to store the token, or to ignore it. The lexical scanner generator then generates a scanner that will try to match a stream of input with the regular expressions. The action associated with the rule that matches the longest string is executed. If two rules match an input string of the same length, the first rule mentioned in the lexer file is the one that matches. We now discuss our tokenization decisions roughly in the order in which they appear in our lexer file.

First of all, we decided to throw away words that contain byte values outside of the ASCII range. We translated the most common wide byte values to their most probable meaning before. Of other wide values we can't be certain what they mean. Rather than adding a big lot of low frequency words we discard words. A word here is defined as a string of non-whitespace characters. Next, we store the few HTML tags that we kept while parsing the html. With two simple regular expressions, we try to match URLs and e-mailadresses, replacing them with two self invented html like tags: `<url/>` and `<email/>`. These tags are nice to keep because they can help us when we try to remove noisy n-grams later.

With these matches out of the way, we concentrated on some rating patterns to filter out. Numbers and floating point numbers were all stored as the token `<number/>`. Not only is this helpful in erasing some rating patterns, it also greatly reduces vocabulary size. Thus, in a machine learning setting, it reduces the number of features. In other words, it reduces the dimensionality of the classification problem. Patterns like A- . . . E+ were ignored. Also single letters B, C, D, and E were ignored. The letter A as a single letter was left in the review. It would appear in any review as a particle anyway and since we did not remove other stop words, why start with the A?

Strings consisting of consecutive asterisk were ignored. Single asterisks were also ignored. Asterisks used to emphasize a single word were stored as a `<b>` tag. Some other punctuation characters that were preserved included '?', '??', '!', '!!', '!?' and '?!'. We will see in chapter 5 that this was not in vain: a question mark appears to be an indication of a negative review. Quotes are a kind of punctuation that require some attention. They may appear in the middle of a word, as in `don't`. We store these words, quotes included. When quotes appear around sentences or words, we keep them. Why? Well, they might signal irony. While this is no serious attempt to detect irony or sarcasm, we tried to keep everything that might contain some interesting signal.

Another interesting phenomenon in English text is hyphenation. Many words exist that may be written as two words concatenated by a hyphen. Examples are co-occurrence and non-trivial. Sometimes two words connected by a hyphen should really be regarded as two words. On the other hand, some words seperated by a space should actually be regarded as one word. The following example illustrates both of these problems. It was taken from an entertaining discussion online [6]:

> the New York-New Haven railroad

Following Manning et al. (2008) we store two strings connected by a hyphen as a single word. When more than two strings are connected by hyphens each string is stored as a word and the hyphens are dropped.

[6]`http://www.hit.uib.no/corpora/1996-1/0227.html`

Finally, to reduce the vocabulary further, we lowercase all words. This does mean that the distinction between proper names and other words becomes more vague. But an additional advantage is that words that are spelled with a capital letter because they occur as the first word in a sentence or as a movie title word are correctly seen as equivalent to their lowercase counterparts. This decision is also a very simple heuristic. In section 5.7 one can see what review 6370.html — our example review in the previous section — looks like after tokenization. In the next section we discuss how we identify n-grams that are likely to be boilerplate.

4.6 Removing noise with suffix arrays

Now that we have tokenized our reviews, they are reduced to a sequence of words. The set of all unique words that appear in our corpus is our vocabulary. If we order the vocabulary alphabetically, we can denote each word with an integer, its sequence number in the vocabulary. This is how we stored the corpus on the hard disk after tokenization: as one large binary file with integers. We inserted the beginning of each file as a separate word in the vocabulary. The whole corpus is now just one large string.

We wish to find in this large string n-grams of longer than a threshold length, that occur at least twice. Why? Because we expect boilerplate to consist of quite long n-grams that appear multiple times. For example, before we saw the review fragment

```
(...) <p> [This
appeared in the 11/14/96 "Bloomington Voice", Bloomington, Indiana.
Michael Redman can be reached at <a> mredman@bvoice.com </a> ] </p>
```

After tokenizing, it would look like this:

```
(...) <p> [ this appeared in the <number/> / <number/> / <number/> "
bloomington voice " , bloomington , indiana . michael redman can be
reached at <a> <email/> </a> ] </p>
```

We would expect several quite long subsequences of this sequence of words to appear often in the corpus, if many reviews that appeared in Bloomington Voice are present in the corpus.

Manber & Myers (1990) propose a datastructure that makes it very easy to locate n-grams above a certain length that occur more than once. This datastructure is a suffix array. A suffix is defined by a position in the corpus string. The array of all suffixes is then a list of integers of the same length as the corpus. These suffixes may be sorted alphabetically. Each suffix has a very useful property: the 'longest common prefix' (lcp) it shares with the suffix sorted just before it.

suffix array

Building a suffix array is a matter of sorting all suffixes. A naive implementation would use a standard sort algorithm on the suffixes. The suffixes should be ordered lexicographically, in the same way that words in a dictionary are ordered. For a typical corpus, this would quickly become impractical, however. By exploiting the fact that sorting results of suffixes can be reused the time complexity of the algorithm can be reduced. We found a C implementation online [7] of an algorithm that builds a suffix array in $O(n\ log\ m)$, where n is the length of the corpus, and m is the longest lcp.

[7] `http://cm.bell-labs.com/cm/cs/who/doug/ssort.c`

Once the suffix array is created, we can walk over it once and store the location of all n-grams of at least length m and occuring at least k times. Such an n-gram is present at position i in the suffix array if and only if the suffixes at positions $i+1, i+2, \ldots i+k-1$ all satisfy the condition $lcp \geq m$. The corresponding entries in the suffix array contain the positions in the corpus where these n-grams occur.

We built a simple interactive program that serves n-grams to a human annotator one at a time. It serves longer n-grams first. If there are multiple n-grams of the same length, it serves the n-grams that occur most often first. The n-gram is printed on the screen, after which the annotator can choose from the following actions:

Ignore Do nothing.

Label as noise Label all occurrences in the corpus of this n-gram as noise.

Label as relevant Label all occurrences of the n-gram as relevant. If substrings of this n-gram are later labeled as noise, that takes precedence. This option is useful because substrings that occur only in n-grams that have already been labeled as relevant will not be served to the annotator again.

Remove files Remove all files that contain this n-gram.

Keep one file Remove all but one files that contain this n-gram. This is useful for duplicate removal. Using this function, we found sixteen duplicates in our labeled reviews. Since after parsing html, we had only 1945 reviews left, after duplicate removal there remain only 1929 reviews. This is the number of labeled reviews with which we did all our experiments.

duplicate removal

Skip This function says: skip subsequent n-grams of the same length. At some point, as occurrence counts become too small, it might not be worthwhile to remove more n-grams of this length. Often, they will contain substrings that occur more often. These substrings will then be served later in the session.

In chapter 5 we present the results of an experiment where we evaluate performance of an SVM on the labeled reviews after a removal session of two hours, where we removed more than twelve percent of all words. Whether we remove noise or not, after tokenization we have a feature vector for each document. These feature vectors may be loaded in the columns of a matrix. This matrix is called the term by document matrix. In the next section we discuss some properties of the term document matrix of our corpus of reviews. The matrix discussed contains the reviews with boilerplate. Only the duplicates were removed.

4.7 The term by document matrix

The result of the preprocessing pipeline or the modeling of the reviews are feature vectors for the documents. These can be combined in a term by document matrix by loading them in the columns. That way, the rows of the matrix can be seen as feature vectors for the terms. The rows then describe each word in terms of its occurrences in the documents. We have seen in chapter 3 that when we calculate an LSA, we are analysing axes of variance of the document vectors and axes of variance of the term vectors at once. A term by document matrix has some interesting properties.

The number of rows corresponds to the number of terms, the vocabulary size. If we want to classify documents using their vector representations, the vocabulary size determines the dimensionality of the problem. In chapter 2 we have discussed why support vector machines can handle high dimensional spaces so well. They do not have to estimate densities. They do not try to model the class conditional distributions in any way. They are solely interested in distinguishing between objects from one class and objects from another class. This is also what makes SVMs so hard to beat. In line with the philosophy behind statistical learning theory they try to solve the easiest problem. The vocabulary size of all 27000+ reviews is 178.833. The vocabulary of the 1929 labeled reviews contains 48.203 words.

In natural language processing, the term by document matrix is typically very sparse. In our matrix, less than one percent of all entries is nonzero. SVMlight is optimized for sparse matrices. That is why preprocessing steps like mean centering the data come at a cost: training time increases from seconds to minutes. Gensim, the software package we use to do the latent semantic analysis, is also optimized for sparse matrices. Mean centering the document vectors caused a dramatic increase in computation time, as we will discuss in section 5.7.

We also note here a striking feature of the classification problem. There are more words than there are documents, which is typical for natural language problems. This means that there are more features than documents. Thus, the space in which the document vectors exist is extremely empty. Imagine how hard it must be to estimate a multidimensional density in such a space. We have discussed how $n+1$ points that are not colinear are always linearly separable in $\mathbb{R}^n$. In section 5.7 we establish that the term by document matrix of our labeled reviews has (almost) full rank. This means the reviews are not colinear. This means the reviews are linearly separable. Thus, a linear classifier can always achieve zero training error. The power of a support vector machine is that by maximizing the margin it greatly reduces the number of hyperplanes that perfectly separate the training points. The direction of the learned hyperplane will "make sense", and the risk of overfitting is reduced.

In the next chapter, we present several experiments. All of these experiments start with a term by document matrix. We built small scripts that can be used to prune this matrix. For example, in inductive machine learning, we first select the training set columns from the matrix. Rows with only zeros are deleted, pruning the vocabulary. Or, rows that have to few non zero entries are deleted. The effect is that the test set is not used to build the vocabulary. Then, the test set columns are selected from the term by document matrix. The rows corresponding to the training set vocabulary are selected from that matrix. In the description of the experiments these preprocessing steps will be described.

Chapter 5

Experiments

In the previous chapters, we have explained support vector machines, latent semantic analysis and semantic orientation. We have also treated the preprocessing of the HTML movie reviews from the (Pang et al., 2002) dataset of 27000+ reviews in detail. This leads to document feature vectors that contain a term count for each word in the vocabulary consisting of all words in the corpus.

This chapter starts with a few small experiments that explore how to normalize the feature vectors before a support vector machine is trained on them. Then, experiments attempting to answer our research questions will be described. At the beginning of each experiment, we will restate the corresponding research question, but now more explicit than in our introduction chapter. In other words, we operationalize our research questions. Then, we describe exactly how we performed the experiment. We end each experiment with an answer to the research question and some discussion. In the last chapter we will discuss the results of all experiments a bit more general, and will discuss possible directions for future research. But before we discuss any experiments, we make some remarks on the basic setup of the experiments.

5.1 Basic setup

For a supervised classifier, the corpus must be divided in a training set and a test set for evaluation. In an inductive machine learning session, it is important that a classifier knows nothing about the test points. This shows itself in the following ways:

- When discarding words that occur too infrequent in the corpus, only the frequency of the words in the training set may be taken into account.
- When the mean input vector has to be calculated, only the training vectors may be used.
- When the standard deviation of each feature is estimated, once again, this has to be done on the training set.
- When a parameter has to be tuned on the training set, no use may be made of the test set.

This may sound obvious, but it is all too easy to sin against this principle. For example, in our first experiment we do a nested cross validation on the training set of each fold to try different values for the hyperparameter C and test how well they generalize. Therefore, in each fold, we divide the training set in k folds again, which gives k training sets. For each of these training sets, we have to preprocess inductively if we want to test how well each of the values for C causes the SVM to generalize.

It is quite difficult to circumvent the problem of fitting parameters to the test set. This is because basically everything from the tokenization to the noise removal to the number of words used to the normalization used to a bug in the code is a parameter. Often one reads that a certain setting 'worked best' (Duin, 1996). In the process of building the total classifier the researcher does a number of test runs and finds early on that some approach just 'doesn't work' that well. In response, a parameter is changed. Over time, performance on the datasets starts to increase. The only way to avoid this pitfall is to do the entire preprocessing on one part of the labeled dataset, test what works and what does not work and only once 'it works' the remaning labeled points may be used. Then, beforehand, it must be designed explicitly which classifiers will be run, and what different parameter settings or parameter tuning methods will be used. At that point, the researcher should push a button once, and then k-fold cross validation results will give performance on truly unseen data.

Also, because everything is a parameter, it is infeasible to search the entire parameter space. One simply cannot compare each classifier on the data of each preprocessing method. That is why in this chapter we work our way from simple to more complex experiments. If, like Drucker et al. (1999), we find that binary features work better for support vector machines than term counts in our first experiment, we will use binary features in subsequent experiments. While this is a form of parameter tuning, our set of labeled data points is not large enough to perform each experiment on a different set. And because there are not that much labeled reviews in the set, it would be hard to generalize the results from the 'pilot set' to the 'experiment set'.

These considerations lead to the question of how the experiments in this research may be compared to existing research. The answer is that they cannot be compared directly using some statistical test. To do this, all preprocessing steps should be exactly the same. However, the experiments may still be valuable. First, we may make a rough comparison showing how performances we obtain compare with existing results. Second, if we find that algorithm A works better than algorithm B, we may compare our results with other experiments in which algorithm A was compared with algorithm B.

For example, suppose we have noted that on the 2000 labeled instances binary features work best, it is best to consider only words that appear in at least five documents, and z-standardization is not worth the effort. Subsequently, we run a support vector machine on this data, and compare its performance with a support vector machine on the same data, but with some additional features. Suppose that the latter classifier performs significantly better. Now, while we can not state that our performance is significantly better or worse than some other researcher achieved on this dataset, we can still make a contribution to research by concluding that the additional features, obtained by latent semantic analysis, for example, are a promising direction for future research. The fact that they worked better despite the fact that the SVM was run under optimal conditions only makes this claim stronger.

5.2 Cross validation and confidence intervals

For evaluation of the classifiers, we chose for k-fold cross validation. Liu & Zsu (2009) treat the pros and cons of different methods of evaluation in a very comprehensible fashion. In k-fold cross validation, the data set is divided in k subsets of equal size, give or take a single instance. Each fold is used once as a test set. The remaining $k-1$ folds are then used as a training set. The k test sets can be seen as independent samples from a much larger population of movie reviews, because they do not overlap. In our research, we must keep in mind that the data set was not randomly sampled from a much larger population. Instead reviews that were not clearly positive or clearly negative were filtered out. In addition, the data set is divided fifty-fifty in positive and negative reviews.

Performance of an algorithm on the population of data points may then be modeled with a normal distribution. The intuition is that the algorithm would achieve a certain true score on the entire population. Differences in performance on independent samples are then attributed to chance, and assumed to vary around the true score. With the k scores of the algorithm on the folds one may estimate the true score, the mean of the normal distribution. The more performance measures one has, the better this estimate will be. Therefore, one would want k to be high. On the other hand, one wants the number of data points in each test set to be high, in order to get a fine grained measurement of performance. A well accepted value for k in machine learning research is 10.

We follow Dehling & Kalma (1995) (in Dutch) in a short explanation of how to estimate the mean performance on a sample of just ten performance measures. Let $X_1, X_2, \ldots, X_n$ be a sample from a $N(\mu, \sigma^2)$ distribution. Then $\overline{X} \sim N(\mu, \sigma^2/n)$. Then

$$\frac{\overline{X}-\mu}{\sqrt{\sigma^2/n}} = \sqrt{n}\frac{\overline{X}-\mu}{\sigma} \sim N(0,1).$$

Since the normal distribution has no analytic primitive function, we have to lookup the values between which lies 95% of the probability mass of $\sqrt{n}\frac{\overline{X}-\mu}{\sigma}$ in a table. These are ± 1.96. Simple rewriting then tells us that with 95% confidence

$$\overline{X} - 1.96\frac{\sigma}{\sqrt{n}} \leq \mu \leq \overline{X} + 1.96\frac{\sigma}{\sqrt{n}}.$$

This is a confidence interval for μ. **confidence interval**

However, it is of limited use, since we do not know σ. If we instead use the sample standard deviation

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{N}(X_i - \overline{X})^2},$$

then $\sqrt{n}\frac{\overline{X}-\mu}{s}$ follows a t-distribution with $n-1$ degrees of freedom, centered around zero as well. Because s is calculated with the aid of $\overline{X}$, only $n-1$ of the observations are independent in the estimation of s. The term degrees of freedom relates to this. This distribution looks like the normal distribution, only it is a little more flat. This causes the confidence interval to be a little wider. Looking in the table for the t-distribution with $n-1$ degrees of freedom, for $n = 10$, we find that 95% of the probability mass is now located between the values ± 2.26. Thus

$$\overline{X} - 2.26\frac{s}{\sqrt{n}} \leq \mu \leq \overline{X} + 2.26\frac{s}{\sqrt{n}}.$$

These are the confidence intervals that are shown in all the figures in this chapter showing the results of the 10-fold cross validation experiments.

In k-fold cross validation, the training sets for all test sets do overlap significantly. Therefore, the k experiments are not completely independent. An alternative would be to randomly sample from a large population a separate training and test set for each repetition of the experiment. In practice, one has to deal with limited data, however.

What conclusions may we draw from the confidence intervals? If the performance of the algorithm indeed follows a normal distribution, then the true mean would be inside the interval in 95% of the cases if the entire 10-fold cross validation was repeated many times. Of course the repetitions would have to be independent. The value of repeating the cross validation on the same dataset is limited, because the test sets would overlap. Ideally, then, each repetition would be done on a freshly sampled set from the population. In this master thesis, we test support vector machines on the same data set under many conditions. We use term features, binary features, LSA features of different dimensions, and combinations of the features. Each change in one of these conditions results in an experiment for which we obtain a confidence interval. Since we run many of these experiments, we should not be surprised if the true mean is sometimes outside of the confidence interval. After all, this is expected one out of twenty times.

Therefore, it is important to relate the results to results obtained by other researchers, preferably on other data sets. We test if binary feature vectors work better than term counts because Drucker et al. (1999) mentioned it in their research on spam categorization. We test if using all terms improves performance because Joachims et al. (1998) found that it did in a text categorization task. Although these task differ from our sentiment analysis task, they use a similar solution. They, too, use the unigram language model, which assumes that from the distribution of the words over the documents it is assumed that a useful classification function can be learned. It is also important to report experiments that did not yield interesting results, instead of reporting only successful ones. In a meta analysis, scientists should then be able to make more reliable inferences.

5.3 Using grid search to find a good value for the cost parameter

One of our research questions is if we can improve accuracy of performance by tuning the hyperparameter C, the cost parameter in soft margin support vector machines. C is called a hyperparameter because support vector machines do not themselves optimize this variable. In (Hsu et al., 2003) one approach is discussed where a grid search is done for C. The idea is that one tries different values for C, and selects the one that gives the most desirable results. Values that they found were 'good to try' are:

$$\{2^{-3},2^{-1},2^{1},2^{3},2^{5},2^{7},2^{9},2^{11},2^{13},2^{15}\}$$

If we automate finding a good value for C, we have created a different classifier. Let us call it C-SVM, for easy reference. We evaluate the performance of C-SVM using k-fold cross validation as usual. How can we let C-SVM determine a good value for C given a training set? Again, we can use cross validation. We split the training set into folds again, say l folds. Now we do a full l-fold cross validation for each value of C. The question we ask ourselves is: which value for C gives good performance on unseen points, in the inductive setting? Therefore, for each of the l folds, when we test on it, we may not use the points in it. Thus, if we want to mean center the points, or scale each feature with its standard deviation, we must estimate mean and standard deviation with the points in the other $l - 1$ folds.

Of course, we may not optimize for C on just one of the k folds, and subsequently use this value for the other $k-1$ folds. Why not? Because then C would have been optimized on the test points for these folds. Even in the transductive setting, this would not be allowed, because the labels of the test points would have been used. Thus, it could very well be that C-SVM uses a different value for C on some of the k main folds!

However, an experiment with these values for C resulted in a flat line. Performance did not vary at all over the ten values drawn from the huge domain $[2^{-3}, 2^{15}]$. In a sample fold, the ten classifiers agreed on 14.3% of the mistakes, while the error rate was 15.89 ± 0.0. What does this tell us? The first thing we should look for is the training error. Indeed, in the same sample fold it is zero for all classifiers. This means that the maximum margin hyperplanes all divide the points in the training set of the fold in the same way, even though the orientation of the hyperplanes may differ slightly. How can this be?

We know that our points will always be linearly separable in this very high dimensional space, since the rank of the term by document matrix is 1928. The only duplicate that remains probably received the same label twice. Remember that one duplicate with opposite labels is enough to make a problem linearly inseparable. These close to two thousand points lie in a very high dimensional space. This means that the space is extremely empty. For all classifiers the hyperplane separates the training points perfectly. It is possible that the hyperplanes vary a bit in orientation, causing training points to fall inside the margin at times, but certainly never on the wrong side of the hyperplane! The small variation explains the occasional disagreement on test points. Drucker et al. (1999) also noted that linearly separable data sets can be very insensitive to the value of C.

SVMlight, when called without specifying a value for C, calculates a value itself, one divided by the average vector length of the document vectors. This value is for our dataset usually much lower than 2^{-3} (on one fold it was 0.0029), and using a lower value actually increases performance. This supports the idea that using slack improves generalization even if a hard margin classifier can seperate the training points perfectly. Since tuning C in the way we did in this experiment is computationally very expensive and not very useful for this problem, we decided to let SVMlight calculate the cost parameter itself in our next experiments. We will see that performances increase. Of course we could decide to run another experiment with a grid search on C, trying lower values, but since we have already had feedback from all the points in our dataset, this would not be a good prediction of generalization on other datasets. We would merely be fitting the current dataset.

5.4 Term counts versus binary features

Drucker et al. (1999) mention that for spam categorization binary features work best, compared to term count features and tf-idf features. Pang et al. (2002) found on the same dataset as the one used in this research, that binary features performed much better for support vector machines. In this section, as a warming up exercise, we describe a simple experiment in which we compare performance on term count features and binary features. Can we verify the result obtained by Pang et al. (2002)?

For this experiment, we let SVMlight calculate a value for C, the cost hyperparameter for soft margin SVMs. We ran SVMlight in an inductive fashion, without showing it the test points. The deletion of low frequency words was done in each fold based on word occurences in the training set. In other words, the preprocessing was done strictly inductive as well. The precise dimensionality of the problem may differ from fold to fold for that reason. The frequency features are just the term counts from the term by document matrix. Binary features for document vectors are one for terms that occur in the document, and zero for other terms. A word is a feature only if it occurs in at least two documents.

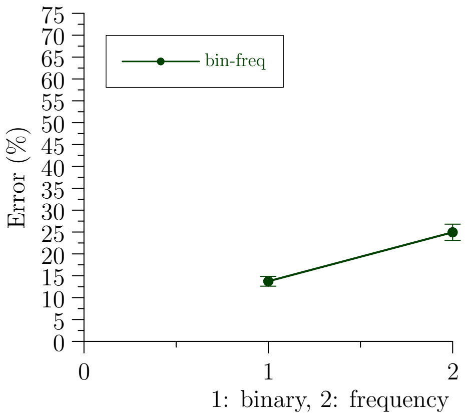


Figure 5.1: Average ten fold cross validation performance of SVMlight on binary features and on term counts. The confidence intervals are 95% confidence intervals from a Student's t-distribution with nine degrees of freedom. Drucker et al. (1999) noted that SVMs worked best with binary features in the bag of word model on the task of spam classification. Based on the large difference between the confidence intervals above one may conclude that this is also true for classifying clearly positive and negative movie reviews.

In Figure 5.1 the average performances of a support vector machine on two different types of input data on our corpus of 1929 labeled reviews are shown. Binary features are compared with term count features. It is clear that binary features perform much better for this task, and significantly so. Why may this be the case? Pang et al. (2002) note that in text categorization, where the task is to classify documents according to topic, certain topic words tend to be repeated. Perhaps this is the case, and word frequency is more important in topic classification. But why the worse performance on sentiment analysis when taking word frequency into account? Perhaps repetition in sentiment analysis can be misleading. For example, when a reviewer lists several positive aspects of a movie but then concludes with one negative remark, which determines his judgement. Another factor could be that we did not use a list of stop words that are deleted from the vocabulary. Certain words have very high frequencies, and could confuse any supervised learner.

5.5 How many words to use

In an interesting experiment, Joachims et al. (1998) rank word features according to their binary information gain. Then, he trains and tests a Naive Bayes classifier discarding the best features. Even if he uses only the worst features, performance is much better than random. Since it seems unlikely that the information in the worst features is completely redundant, he concludes that in text categorization there are few irrelevant features. This is a possible explanation for the fact that support vector machines perform well in these tasks.

In this experiment, we explore how the number of words used influences performance, by discarding words that occur in the smallest number of documents. This is perhaps the most simple form of feature selection imaginable. Preprocessing and running were done inductively, as in our previous experiment. We used $C = 1000000.0$. In effect, then, the SVM runs in hard margin mode, since the cost of misclassifying a training instance is way higher than any gain in margin that could be achieved. In the case that the data points are not linearly separable SVMlight would terminate training and issue a warning. But we noted already that because there are more dimensions than training instances, the points will always be linearly separable. The only exception would be if the points would actually lie in a subspace of a dimension lower than the number of training points, which would be very unlikely. We only varied the minimum number of documents a word must occur in to be used as a feature.

In Figure 5.2 the results of the experiments are shown. There is no significant difference in performance between any two conditions. Apparently, in the task of sentiment analysis, it is hard to gain performance increase by using more low frequency words. Another way to interpret the results is that support vector machines can handle many features well. Performance does not decrease as dimensionality increases. The results also indicate that because of the high dimensionality of this type of problem, hard margin SVMs may be used with succes.

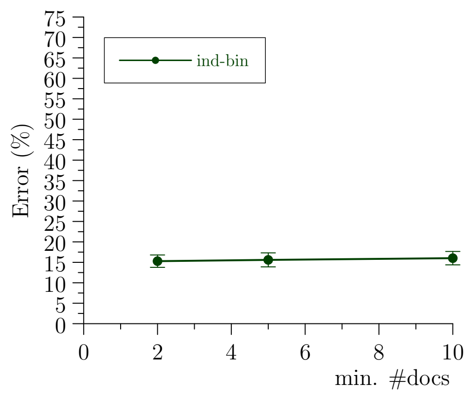


Figure 5.2: The performance of SVMlight on binary features for words that occur in at least two, at least five or at least ten documents. There is no significant decrease in performance as low frequency words are dropped.

5.6 Noise removal or not?

This experiment attempts to answer the research question: does noise removal improve review classification? Or almost equivalently, does not removing noise degrade performance? In chapter 4 we described our application that offers n-grams to a human annotator one by one. Long n-grams are offered first and if two n-grams have equal length, n-grams that occur more often in the corpus are served first.

For this experiment, one noise removal session that lasted two hours was done. Slightly more than 12% of the words in the corpus of labeled reviews were removed. In the midst of boilerplate such as "See this and this URL for more reviews" often author names were also removed. Author names may be important features for a classifier, however. Another session where more care is taken to avoid deleting author names might produce slightly better results.

For both the corpus with and the corpus without noise, words that appeared in less than three documents were removed from the vocabulary. Because SVM performance proved so robust with regard to the number of words that are used in the previous experiments, we just used the entire datasets for this reduction of vocabulary. Strictly speaking, these experiments are transductive, because the test documents are also used for the vocabulary pruning. However, the difference between this and pruning the vocabulary in each fold of a cross validation based on the training set is negligable. Binary features were used.

A 10-fold cross validation on this corpus did give the best performance in this research. On the corpus where noise was removed an error rate of 12.96 ± 1.80 was achieved. Compared to 13.89 ± 1.68 on the corpus with noise this is not a significant improvement, however. Joachims et al. (1998) concludes that 'With their ability to generalize well in high dimensional feature spaces, SVMs eliminate the need for feature selection, making the application of text categorization considerably easier.' This experiment seems to back up this claim. A two hour noise removal session with the suffix array based n-gram server did not result in feature vectors that were significantly easier to classify. In all other experiments we used the corpus where no noise was removed.

5.7 A latent semantic analysis of 27000 reviews

Now that we have seen that SVMs have very strong performance on this task of separating clearly positive from clearly negative examples, comparable with the results Kennedy & Inkpen (2006) obtained on this set, we set up some experiments that try to use the 'topic' information we gather from a principal component analysis on the entire dataset of more than 27000 reviews. In this section, we discuss the latent semantic analysis itself. Note that the terms LSA and PCA are sometimes deliberately intermingled, because of their close relationship detailed in chapter 3. We show what the points projected on the first two principal components look like. We show a scree plot of the singular values, and we show some sample topics that are found. Finally, we visualize what happens to a review when it is projected on the subspace spanned by the 1250 principal components.

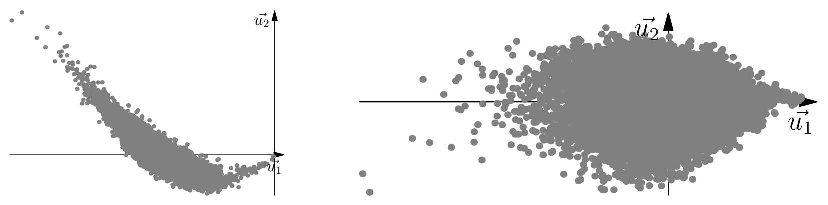
Before doing the analysis, words that occur in less than five documents were removed from the vocabulary, resulting in a vocabulary of over 50.000 words. This reduces computation time, and memory requirements for the U matrix in the SVD, which has the same number of rows as there are words. The 1250 left singular vectors corresponding to the 1250 highest singular values were kept. For calculating the SVD, Gensim version 0.5 [1] was used. It implements a relatively new algorithm detailed in (Brand, 2006). While calculating the SVD, it does not have to keep the entire corpus in memory, which minimizes the internal memory required for the computation. To reduce the memory requirements further Gensim does not store the matrix V in the SVD USV^T by default. Because we use only document similarities, even in our SO approach detailed in section 3.8, we also discarded this matrix with right singular vectors.

The success of binary features over term frequency features inspires to try a latent semantic analysis on the binary term by document matrix as well. Two analyses were done. One on the binary features. Computation on a machine with a 2.2GHz PC lasted more than half a day. A second on mean centered binary features. Mean centering the binary document vectors means that the term by document matrix is no longer sparse. Because Gensim is optimized for sparse matrices, computation lasted considerably longer, well over four days. The time complexity of the algorithm in (Brand, 2006) is $O(mnk)$ where m is the number of terms, n is the number of documents, and k is the number of singular values that one wishes to keep.

When we do an analysis on data that was not mean centered, we expect the first left singular vector to point in the direction of the mean (or in the exact opposite direction). What are words that have high coefficients in this vector? Of course, they are the most frequent words. In a small latent semantic analysis done on the 1929 reviews for which we have labels, with 1250 dimensions, and done on words that occur in at least 20 documents (around 8000 words), the ten words with the highest coefficients were: '.', a, of, the, and, ',', to, in, is, that.

We also did a small PCA on just the 1929 labeled reviews. This allowed us to calculate the full SVD, with 1929 topics. For this PCA, we kept words that occurred in at least three documents: around twenty thousand words. The curve of singular values looks just like the curve in Figure 5.4 below, but the

[1] `http://nlp.fi.muni.cz/projekty/gensim/`



(a) Projection on the first two left singular vectors

(b) Mean centered data projected on two left singular vectors

Figure 5.3: Figure 5.3a: 27000 reviews projected on the first two left singular vectors found by an LSA on the binary document vectors. The first vector points away from the mean. Figure 5.3b: An LSA on the 27000 binary but mean centered documents data is like a PCA. The features are uncorrelated and capture the directions of maximum variance.

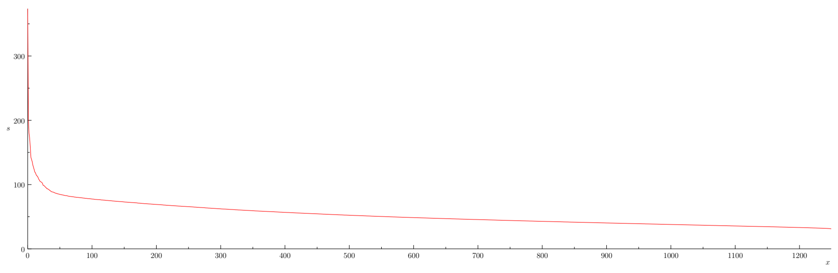


Figure 5.4: The singular values of the LSA on 27000+ reviews, ranked in descending order. A plot like this is called a scree plot.

very last value dropped suddenly to zero. Apparently the 1929 reviews lie in a 1928 dimensional space. In other words, the rank of the term by document matrix of the labeled reviews is 1928. The most likely explanation for this is that still one duplicate review slipped by the duplicate removal preprocessing step.

Now we will take a look at a difference between latent semantic analysis done on binary features or on mean centered features. In Figure 5.3a we see the 27000+ reviews as binary document vectors projected on the first two principal components. The features in this two dimensional space seem negatively correlated. A latent semantic analysis on mean centered data is like a principal component analysis. The projections of the mean centered binary document vectors on the first two left singular vectors in Figure 5.3b are clearly uncorrelated.

scree plot

The singular values ranked from high to low of an LSA on 27000 mean centered review document vectors show an interesting decay, as can be seen in Figure 5.4. In a PCA, it is common to plot the eigenvalues of the sample covariance matrix in this way. The singular values are proportional to the square root of these eigenvalues. A plot like this is often called a scree plot, because its shape resembles debris lying at the side of a mountain. The debris begins where the plot suddenly stops descending fast. This point is also called the elbow in the graph. Often only the number of principal components corresponding to the elbow is kept for the analysis. See (Zhu & Ghodsi, 2006) for an overview

	positive	negative
$\vec{u_1}$	granger's granger gauge susan grade null sub-par unquestionably byte bytes	we even way do them what had see because been
$\vec{u_2}$	i my you me ? don't your think bad really	director screenplay running cast its life young between while james

Table 5.1: Words with highest positive and negative coefficients in the first two topics of the LSA on 27000+ mean centered movie review document vectors. Notice that the negative components were larger for both singular vectors. Topics found by an LSA can be hard to interpret.

and an approach to locate the elbow in the scree plot automatically.

What are the topics that a latent semantic analysis or a principal component analysis finds? We have noted in chapter 3 that when binary features are used, features that vary most about their mean are words that occur in half of the corpus. It could be that these features have high coefficients in the top principal component. Consecutive principal components will be orthogonal to earlier components, however. In a PCA, we have that the marginal distributions along the principal components are all uncorrelated (Zhu & Ghodsi, 2006). With all this knowledge, can we make sense of some sample topics?

It is often noted that the topics that an LSA finds are hard to interpret (Manning & Schütze, 2000). This is true. Table 5.1 lists the most positive and most negative words appearing in the first two singular vectors. Words that appear in a singular vector with negative coefficients can be interpreted as causing a review to score low on the 'topic' of the singular vector. Of course, words that appear with positive coefficients would then cause a review to score high on that topic. Looking at the words in table 5.1 it is hard to distill any conceptual topic from the words.

Can we visualize what happens to a review when it is projected on the subspace spanned by the principal components? Remember from chapter 3 that the projection is accomplished by the matrix multiplication with the aligner matrix U^T. All we have to do to see what happened in terms of the original axes, the words, is multiply the projected vectors with the hanger matrix U which has the principal components in its columns. Because the points had been mean centered before the projection, we translate them back with the same mean document vector after multiplication with U^T. First, we show the words in the review before the projection. These words are from a randomly chosen review, review 6370.html:

> **a about accomplished acting action actually adequate again agrees airline alive all all-american also although an and anti-hero apart apartment appeared are arguing around as at back balcony bars be been before begin being below best big bloomington brawley break build builds business businessman but buy by calls can candy career case casting character characters coke continues copyright course cowboy crafted danger delroy did differently do doesn driven**

> **dwell each effective elements eloi em emotionally especially even explores fair fall falling fate fbi feeding few film filmed first five flaws for forget frolic from g gary gazillionaire genre get gibson going good got gritty guns h hand hard-nosed has have he here hero high him his home hour howard i if ignore in indiana into is it junior just kidnapped kidnappers kids known large later latest leader leave lindo lowlifes mad man mark mastermind max mel michael minute moments most movie much mullen negotiations new nicest nolte not now numerous obviously occasionally occupying of off on once one order other out over owner pay penthouse people perfect personalities play plays powerful previous problems progresses provide puts ransom rather reached real realizes redman rene representations resolution reveal review right ron rough round russo s satisfying says scene scheming science sean seems seen several shaken shed shoot sinise situation skin snatched society some someone son space speech still story tacked tactics talk talkie team tend than that the their there these they things this to tom too top trio trouble tycoon ultimate unsuspecting up upending veteran vision voice want watch way we well wells weren what when while who whose wife willing with won wonderfully work working world wouldn you**

Next, we show which words are present in the review that has been 'dipped' in the 1250 dimensional space. Words are displayed in three different colors. Black, if the term weight is closest to one. Gray, if the term weight is closest to 0.67. Light-gray if the term weight is closest to 0.33. If the coefficient is closest to zero, the word is omitted.

> **a about acting action actually again agent agrees alive all also although an and apart apartment appeared archive are around as aspect at award back be been before begin being below best big bloomington break build building builds business but buy by calls can career case casting character characters classics column connery contacted continue continues cop copyright course crafted crowd danger decision delivers delroy desperate did directing do don door driven each effective elements email emotionally especially even exciting fair fall falling fate fbi feelings few fiction film filmed first five flaws for forget from g gary genre get gibson going good got h hand hanks has have he henry here hero high him his home hour howard i if in independent indiana individual intense into is issue it just kidnapped kids known large later latest leader leading leave lindo mad man mark max mel michael million minute moments most movie much new nick not now numerous obviously occasionally of off on once one order other out over owner pay people perfect personalities play plays powerful previous price problems provide purpose puts ransom rather reached**

> **real realizes redman rene resolution review right ron rough rush russo s satisfying says scene science sean seems seen sent several shoot sinise situation society some someone son space speech stands still story straight strength t talk team tend than that the their there these they thin things this to tom too top tough trouble ultimate up veteran villain vision voice want watch way we weapon week's well western what when while who whose wife william willing with won wonderfully work working world you you've**

We display all words just to show at least one example of what happens to a review. And, this is fascinating material to look at. In the projection, there are 270 words shown. In the original, 278 words were present. Quite some words get downweighted a lot. For example, the word 'gun' is present in the review, but in the projection the word weight is closest to zero and the word is not shown. However, in the projection there is the word 'weapon' in dark grey which is not present in the original. Glancing over some of the added words, they appear to be very similar to words present in the original. We have 'you've' added to 'you'; 'continue' added to 'continues'; 'hanks' added to 'tom'; 'connery' added to 'sean'. Which words are lost? Many of them are rare words, such as 'frolic', 'mastermind', 'unsuspecting'. Indeed, reading the projection, one can't escape the feeling that common words have stronger word weights.

Now that we have given some results of the latent semantic analysis itself, we are curious how it will behave when it is used in combination with a supervised classifier such as an SVM. In the next few sections, we will study different ways of using document concept spaces of various dimensions as calculated by an LSA on the binary feature vectors or the mean centered binary feature vectors.

5.8 Projecting documents on their principal components

In this experiment, we will apply latent semantic analysis as a preprocessing step. Before classification, it is not uncommon to do a principal component analysis as a form of normalization or dimensionality reduction. We compare the performance of LSA on binary features with an LSA on mean centered binary features. For both techniques we varied the dimension of the document concept space on which the documents were projected. This is very easy once the semantic analysis for the 1250 topics has been done. To get the vectors projected on a space of a smaller dimension, one can just disregard the last feature values.

The experiment allows us to answer some of our explorative research questions. First, does compressing the feature vectors with LSA in this matter affect performance? Does mean centering in advance make any difference? And finally, how does performance behave when we vary the number of principal components we keep?

Since we have already used all reviews for the LSA, including the labeled reviews from the test set, this experiment is a transductive machine learning experiment. No additional preprocessing is done for any fold. SVMlight was run in an inductive fashion. It was not presented with any test points during training. By default, SVMlight calculates its own value for C, we let it do that for this experiment, too.

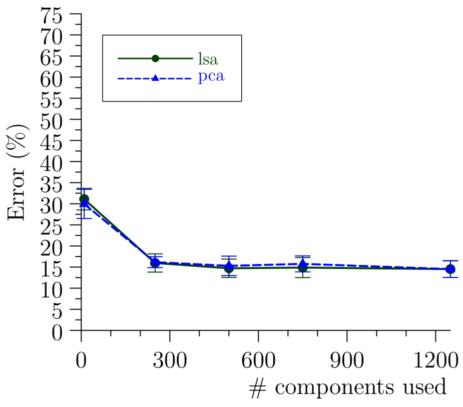


Figure 5.5: Using LSA as a dimensionality reduction technique. In this case, the review document vectors are projected on the top k left singular vectors. This is the space in which document similarities can be calculated with the low dimensional projections. Mean centering the data in advance gains no advantage over omitting it.

In Figure 5.5 the results of projecting the documents on spaces of 10, 250, 500, 750 or 1250 left singular vectors are shown. Even with only 250 dimensions, performance is on par with the performance on the original binary feature vectors in the previous experiments. Using more dimensions does not decrease or increase performance significantly. Mean centering the data in advance gains no advantage over omitting it. In section 3.5 we mentioned already that this is probably due to the fact that in high dimensional space there are enough degrees of freedom left for the remaining axes to capture the variance. Only when one tries to interpret the topics one should be aware that the first topic is simply in the direction of the mean if the data was not mean centered. The result of this experiment suggests that mean centering is not necessary as a preprocessing step when LSA is used for classification. Because the matrix remains sparse without mean centering, calculating the LSA is much faster.

An interesting additional question is: is it overkill to use the 27000+ reviews for calculating the left singular vectors when they are used in this way? Turney & Littman (2003) found that for unsupervised classification of words as either positive or negative, it was of crucial importance that the LSA was done on a large corpus. In the supervised setting of this experiment, what would performance be on an LSA calculated just on the labeled reviews? A 10-fold cross validation on the small PCA that we calculated on just the labeled reviews, where we kept all the 1929 topics, shows a $14\% \pm 1.87$ error rate. What then, is the advantage of using the whole corpus of reviews? Let us see how performance declines if we use less dimensions. Perhaps the directions that were calculated using the entire corpus capture more important information?

In Figure 5.6 the results are shown of an experiment with a PCA done on only the 1929 labeled reviews. The number of principal components kept is varied. Performance varies in the same way as it did with the PCA that was computed on the larger corpus of 27000 reviews. When LSA is used as a preprocessing

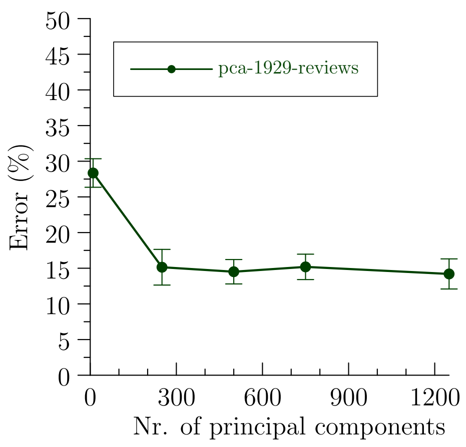


Figure 5.6: The results of keeping 10, 250, 500, 750 or 1250 principal components of a PCA done on only the 1929 labeled reviews. Only when just ten components are retained does performance decline significantly. These results suggest that when data is preprocessed with PCA before classification with an SVM, it offers no benefit to calculate the PCA on a large corpus.

step on the document vectors before running an SVM, it seems that there is no reason to do the latent semantic analysis on a larger corpus.

5.9 Reducing the amount of training data

This experiment will try to answer our research question: does transductive machine learning improve performance? And we investigate what happens when we reduce the amount of training data used. In each fold the training points used were selected randomly from the complete training set. This improves the validity of k-fold cross validation somewhat, because if less training points are used, then over the folds the training sets will become less dependent. Because the k-fold cross validation was not done stratified, no care was taken to distribute the classes over the folds evenly. The transductive algorithm of SVMlight described in chapter 2 thus has to estimate the proportion of positive reviews from the training set.

Finally, we mention some differences between inductive and transductive preprocessing. In the transductive setting, we always only use words that are present in both the training and the test set. This ensures that we never use words that are guaranteed to be irrelevant. In the transductive setting, when we estimate mean and standard deviation, we always do it on the entire set currently under consideration. So if we are tuning C on one of the k main folds, we only estimate them once for each fold, and once for the entire corpus. All experiments that use latent semantic analysis are transductive machine learning experiments, because the analysis was done on the entire corpus of 27000+ reviews, many of which are neither positive nor negative, but all of which are movie reviews. With this in mind, we will discuss the experiments one by one in the sections below.

In Figure 5.7, we see the results of an experiment in which we compare

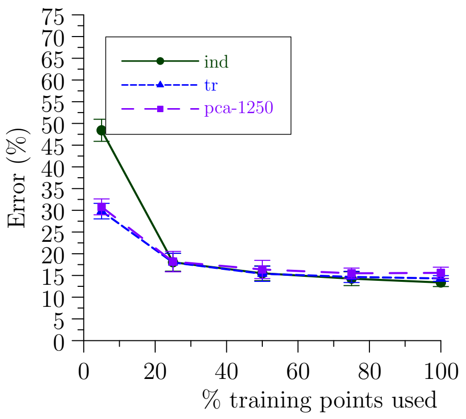


Figure 5.7: Performance of the transductive SVM algorithm implemented in SVMlight compared with the standard inductive algorithm. The transductive algorithm was run on the binary features, or on the 1250 top principal components found by the LSA on mean centered data. Only when 5% of the training data is used does the advantage of the transductive algorithm become apparent.

three different algorithms using 100%, 75%, 50%, 25% or 5% of the training data. The transductive algorithm implemented in SVMlight and described in Joachims (1999b) was evaluated on the binary features (a word was used as a feature if it appeared in at least two documents) and on the top 1250 principal components found by the LSA on mean centered data. When SVMlight is not given any unlabeled points in the training set, it implements a soft margin SVM classifier. The inductive algorithm is here evaluated on the binary features, again using only words that appeared in at least two documents. Performance is nowhere significantly different. Only when 5% of the training data is used does the transductive algorithm clearly outperform the inductive one. However, performance detoriarates also for the transductive algorithm. In the remaining experiments, we always use 100% of the training data. Since the transductive algorithm shows no advantage in this case, we use the inductive algorithm anywhere, even where preprocessing was done in a transductive fashion.

Before we move on to more experiments, we will take a look here at what it is that a support vector machine with a linear kernel learns. Why here? We have compared three different algorithms on five different percentages of training data. That is a total of fifteen different classifiers. Which reviews are misclassfied by all classifiers? This will give us some insight in what are the hard problems for support vector machines on the bag of words data. Also, it will be interesting to see what are the distinguishing directions in the LSA document space.

In Table 5.2 we can see the top 20 positive and the top 20 negative words that SVMlight finds on one of the k folds in the cross validation. Binary features were used, and 100% of the training data was used. We explained in chapter 2 how the direction of $\vec{w}$ may be used to see what were the most important features in retrospect. Since $\vec{w}$ points in the direction of the positive class, features that appear with high coefficients in the vector contribute most to the decision to classify a test review as positive. Similarly, the strongest negative

positive	**negative**
hilarious fun life most also terrific others very excellent great especially well performances makes perfectly true sometimes deserves works back	bad plot worst nothing supposed unfortunately only boring ? poor script attempt awful have waste ridiculous stupid flat reason looks

Table 5.2: The direction of the normal $\vec{w}$ of the maximum margin hyperplane that the SVM finds can be used to rank the features from most positive to most negative. Above the top 20 positive and the top 20 negative words are listed that SVMlight found in a random fold on binary features using all training data. The words that pop up can be an inspiration for new hypothesis generation. For example, why do we find the word 'also' in the positive features? And why does the question mark appear in the negative words? And what about intuitively neutral words like performances, plot and script?

coefficients belong to the features that contribute most to a negative decision.

Some remarks have to be made on interpreting $\vec{w}$ in this way. It is important that each feature has been treated in the same way, and scaled in the same way. If one feature was measured in centimeters and another in meters, for example, the results would be very different. Also, the distribution of the points in the space may be important. If words have very little variance in their usage, because they hardly occur, or because they almost always occur, they will have a small effect on the classification decision even if their coefficient in $\vec{w}$ is strongly positive or negative. A glance at the words in Table 5.2 reveals that ranking the words in terms of the coefficients of $\vec{w}$ is sensible. It would be an interesting research question what kind of conclusions may be drawn from it, if any.

	positive	**negative**
topics	24 41 44 31 8 46 22 353 21 247 39 11 545 97 698 390 55 707 241 389	19 15 1 34 9 4 10 5 27 49 36 35 254 847 2 156 360 225 33 43
words	hilarious makes especially life true back most light others great sometimes fun wonderful pace also something excellent credit performances easy	worst plot only nothing bad poor have boring ? supposed attempt unfortunately reason dull falls worse could name fails potential

Table 5.3: The direction of $\vec{w}$ in the document concept space spanned by the 1250 first left singular vectors can be used to see which 'topics' contribute most to a positive or negative classification decision. The lowest numbered topics in the table are the ones that capture most variance. By projecting $\vec{w}$ back on the original space with the hanger matrix U from the SVD, we can once again see which words were important. This data came from the same fold as the data in Table 5.2 above. One can see the striking similarity in the function that is learned.

Can we also find out what the top 20 positive and negative features are that SVMlight finds on, say, the 1250 dimensional space? Sure, we can find out which topics are most important. Since the topics are themselves linear combinations of words, it should be possible to find which words contributed most. The simplest way of doing this is express the vector $\vec{w}$ itself on the original space using the left singular vectors loaded as columns in a matrix U. In other words, using the matrix U that we found in the SVD: $M = USV^T$. U is a hanger matrix, and can be used to express points that are expressed in terms of its columns in terms of the original axes again.

In Table 5.3 we do this for the same fold in the cross validation that was used in Table 5.2. Lower numbered topics correspond to singular vectors that capture more variance. It is an interesting result that the two top singular vectors appear as important negative features. On average, important negative features are singular vectors that capture more variance than the singular vectors that contribute most to a positive decision. Why could this be? We have noted in section 2.9 that with binary features the words that vary most about their mean are words that appear in half of the corpus. In this corpus, we know that half of the reviews is negative, and half is positive. Could it be that negative vocabulary is more limited? If so, then one would expect negative words to appear in more negative reviews. These words then have more variance across the corpus. This in turn could lead to higher coefficients in the directions of principal covariance.

If we project the normal $\vec{w}$ of the maximum margin hyperplane in the 1250-dimensional space back on the original binary feature space, we see that the function learned by the support vector machine is strikingly similar to the one learned by SVMlight on the binary features. We have seen in an earlier experiment that performance of an SVM remains the same even if only 250 principal components are kept. In this experiment, PCA features also do not seem to differ significantly in performance for any amount of training data. Note that the PCA experiment was also run transductively: SVMlight was shown the test points. We have seen that the same words are important for the decision. We may therefore expect an SVM to make the same mistakes on PCA features as it makes on the binary features that were used to compute the PCA. If we compare the mistakes that are made when all training instances are used between the PCA features (transductive) and the binary features (inductive), we find that 222 of the 1929 points (11.5%) are misclassified by both algorithms. PCA is better on 36 points, but worse on 79 points. Indeed the largest part of both their mistakes are the same.

How many reviews are misclassified by all fifteen classifiers in this experiment? If their mistakes were uncorrelated we would expect the chance that any review would be misclassified by all classifiers to be negligable. However, seeing how all classifiers score easily into the mid eighty performance range on the unigram word model, but not easily above 86%, one gets the feeling that the mistakes of these classifiers must be correlated. Indeed, 2.43% of the reviews is misclassified by all fifteen classifiers. We quote from one of these reviews, randomly selected (14196.html):

> Lastly, Armageddon is out there to save the PLANET, with a capital "P". This can't be any old thriller–we must have an asteroid, the size of Texas head straight for earth!! And, and, we must have beautiful scenic worldwide shots, like Paris, BLOWN UP. And, and, we must have peoples of all colors, nations, and religions, join hand in hand

> for one final, hopeful, HUG. The final sequence, where Muslims in prostrate worship stand up–in Domino fashion–to cheer the victory, filled me with such emotional goo that I wanted to, to, to rip the screen into shreds!! mean, I mean, cheer for ecstatic joy! (It was over).

This quote is dripping with sarcasm. With a unigram language model, irony is hard to detect. Here we have the words beatiful, scenic, hopeful and even ecstatic joy, yet for the human reader the review is clearly negative. In the rest of the review negative words are also to be found, but in the linear classifiers in this research it is really a matter of which kind of words appear most. To solve this kind of problem, one would need to detect irony and sarcasm. Tsur et al. (2010) describe a semi supervised method to recognize sarcastic sentences.

Let us quote from another random review, review 7791.html:

> The critical difference between THE ROCK and CON AIR is Cage's performance. In THE ROCK, Cage appeared to be having the time of his life jumping into the action fray for the first time. (...) No one who loathed THE ROCK for its swear-grunt-blast repetitiveness will be converted by CON AIR; no one who loved THE ROCK for exactly the same reasons will be dissuaded

Perhaps in this review a main problem for our classifier is that there is much comparison with another movie. For clarity, this review is about the movie Con Air. To solve this type of problem one would have to parse sentences and reason about to what object or concept in the real world a phrase is referring. This is a very hard problem.

We will conclude this section with a quote from one more random hard review: review 20092.html.

> eXisTenZ has all the Cronenberg-gore that is expected of him. Even so, this film is no-where near his cult classics such as Videodrome or Scanners. Not even half as suspenseful as the commercially successful The Fly. eXisTenz can probably be looked upon as his 90's version of Videodrome, even so, it is a poor follow-up. While in most of his famed films, his penchant for gore always hit the right note with theme of the film and plot. In eXisTenz, the gory sequences are no more attached to spirit of the film and seem to be an act of over-indulgence than anything else.

In this review, one of the problems is the same as in the previous example: the comparison with other movies muddies the water.

5.10 Adding principal components to binary feature vectors

Since the principal components are often referred to as topics in latent semantic analysis literature and we are interested in studying interaction between topic of a movie and what words are discriminating between positive and negative reviews, why not just add the top k principal components to the binary feature vectors? All words were used that occur at least once in one of the labeled reviews. Then, to these binary feature vectors either the top 10, 25, 50, 100 or 250 were added.

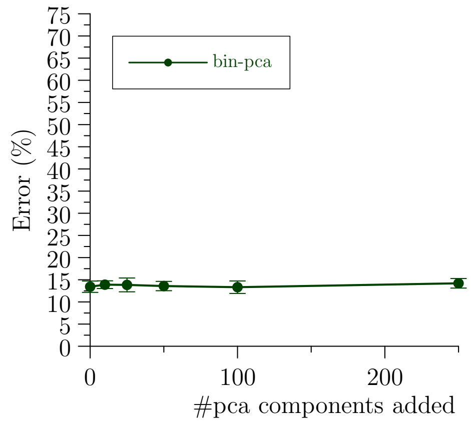


Figure 5.8: Performance of SVMlight on the full binary feature vectors with top principal components added. Either 10, 25, 50, 100 or 250 components were added. No apparent difference in performance is to be noticed. This is not surprising, because the topics are really just linear combinations of the binary features.

In Figure 5.8 we see that performance does not vary at all over these conditions. We discussed already in chapter 3 why this is to be expected. The topics that an LSA finds are just linear combinations of the binary features in the original vectors. Therefore, the points will not in reality lie in a higher dimensional space afterwards. Linearly inseparable points will remain linearly inseparable. But also, linearly separable points will likely not become 'better linearly separable', or linearly separable with a higher margin and better generalization. Another way to see the problem is that the topics really don't add any new information. The SVM can learn the optimal linear combination to distinguish a positive review from a negative one. Adding features that are just other linear combinations does not change anything.

5.11 Adding semantic orientation to binary feature vectors

As outlined in chapter 3, to calculate semantic orientation between a review and a reference set of words, we calculate their similarity in document concept space. Instead of just using *pwords* and *nwords*, we also added a dictionary loosely inspired on emotion research. We did not use only bipolar sets of words, but consider LSA similarity to each set a feature. The cosine similarity was chosen as a similarity measure because the length of the folded in documents is much smaller than that of a typical review. The cosine of the angle between two vectors equals the dot product of the vectors scaled to unit length, so length is ignored.

It turns out, the approach didn't work at all. Performance was hardly above chance level. Could it be that the quality of our lexicon was poor? Instead

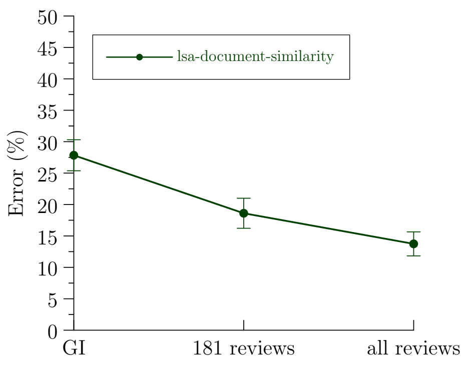


Figure 5.9: Performance of SVMlight on three kinds of features: (1) cosine similarities to 181 GI categories projected as pseudo-documents on 1250 left singular vectors. Performance is above chance level. Are these good features? For comparison, (2) cosine similarities to 181 random reviews of the dataset were used, leading to much better results. A natural question is: what happens if (3) the cosine similarities to all reviews are used? The result is on par with our best results in this research.

of the hand crafted lexicon we used all 181 categories of the General Inquirer. See Figure 5.9 for the results of this experiment. The GI categories were also folded in in the document concept space computed on all 27000 reviews. Cosine similarity with these documents improved performance to 27.84±2.47 error. But when these features were added to the standard binary features performance did not increase. Theoretically, distance to prototypical or interesting data points can make the points better separable. Maybe these points are not so interesting at all for classifying reviews as positive or negative? As a reality check, the cosine similarity to 181 random reviews was used instead of the cosine similarity to the 181 GI categories. Performance increased dramatically to 18.61 ± 2.39. Certainly this is no encouraging news for the usefulness of the 181 GI projected points. A natural follow up question is: what will the result be if the cosine similarity to all reviews is used instead? With 13.47 ± 1.91 this is on par with the best results in this research.

Chapter 6

Conclusions and future work

In the chapter on the experiments, we already did some discussion of the results. In this chapter, we draw more general conclusions from all the experiments together. We conclude with a section on future work.

6.1 Conclusions

6.1.1 A ceiling effect for the bag of words model

Three main approaches in our experiments reached the mid eighty performance level that Kennedy & Inkpen (2006) also reached. First, the experiment on binary features where SVMlight was allowed to set its own value for the cost parameter. Second, when first compressing the document vectors with PCA or LSA. Even if only 250 dimensions were kept performance did not change significantly. Third, when cosine similarity in the PCA compressed space to all reviews was used as a feature vector. Our first conclusion is the impression that a ceiling has been reached in performance on this task with the unigram language model.

6.1.2 Why LSA compression does not change SVM performance

In our introduction to LSA, we conclude that it resembles PCA very closely. This is not a novelty, it is well known. The reason that we discovered just how similar the techniques are when we already decided on LSA is that in information retrieval literature we consulted PCA was rarely mentioned. This is a historical matter, the names latent semantic indexing and latent semantic analysis are just more popular.

It is also well known that the dimensions that capture the most variance are not always the most discriminating features for classification. Shima et al. (2004) cite an example in face recognition research where recognition improved after removing the first three principal components. It did not come as a surprise that the compression to a space of anywhere between 250 and 1250 principal components did not improve performance significantly. A bit more interesting is that there was no difference at all between the tried values. The fact that there was even no difference when using the principal components of the smaller corpus raises an interesting question: "Does it matter at all what the direction of the components is on which the data points are projected?".

In fact, there is a theorem about this, the Johnson-Lindenstrauss lemma. Sahlgren (2005) introduces Random Indexing, one of several dimensionality reduction techniques that is related to latent semantic analysis, but rests on the Johnson-Lindenstrauss lemma, which states that "if we project points in a vector space into a randomly selected subspace of sufficiently high dimensionality, the distances between the points are approximately preserved." This could mean that when compression is done as a preprocessing step before using a support vector machine, it does not really matter what subspace is used. If the distances between the points are preserved, an SVM should be able to find perhaps the same support vectors that separate the points best. To work this out, further experiments would be needed. We did show in our experiments with the simple trick of expressing $\vec{w}$ in terms of the original term features again that the support vector machine learns a very similar function.

For many machine learning algorithms dimensionality reduction should be interesting. This is because density estimation becomes easier as the lower dimensional space will be more dense. Naive Bayes does marginal density estimation. K-nearest neighbour can even be interpreted as a form of density estimation. However, support vector machines with a linear kernel do no density estimation. Period. In chapter 2 we discussed statistical learning theory. An underlying principle is that you should always solve the easiest problem. Never solve a harder problem, e.g. density estimation, that will give you the answer to the easy problem: classification as a side effect (Vapnik, 1998).

6.1.3 Semantic orientation or similarity to other reviews?

An interesting result is the strong result achieved with LSA cosine similarity to 181 random reviews, compared to the 181 GI categories. Using cosine similarity to all random reviews resulted in a performance on par with our best results. As we discussed in the related work section, Kennedy & Inkpen (2006) used an ensemble machine learning algorithm that combines the unigram SVM approach with a kind of semantic orientation by counting positive and negative terms that were obtained from the General Inquirer lexicon (GI). It did not improve performance significantly. Only when in addition sentences were parsed and negation and intensifiers were modeled with special bigrams the ensemble did achieve a small but significant improvement of 1.3 percentage points.

It could be interesting to try an ensemble with our GI based features, too. However, since the 181 GI categories work much better, why not make an ensemble with these features? Using cosine similarity among the reviews themselves is similar to using a radial basis function as a kernel. Yu et al. (2008) try an ensemble of several support vector machines, each trained on the same data but with a different kernel. No definite conclusion is reached.

6.2 Future work

6.2.1 Aggregrating over multiple reviews

While a ceiling seems to have been reached for the unigram language model in predicting the sentiment of a single review, predicting the volume of positive or negative reviews in a set might yield better results. This is a common principal in statistics. Mishne (2005) tried to predict emotion tags that people associated with their own blog posts on LiveJournal.com. This proved to be a very difficult task. But when they subsequently investigated if the volume of blog posts with certain emotion tags at a certain hour could be predicted (Mishne & de Rijke, 2006), they achieved very good results. In movie reviews, an interesting task could be to guess the average grade a movie receives based on reading in all its reviews. Since IMDB makes available how often each grade from one to ten is given to a movie, it would also be interesting to predict the entire grade distribution.

6.2.2 Weighing principal factors

When projecting the hand crafted emotional lexicon documents on the first two principal components to get a feel for how the algorithm would work, it became clear that documents containing words of opposite meaning were actually close to each other. So for instance *pwords* and *nwords* described in chapter 3 would lie almost on top of each other in Figure 5.3b. A natural idea is that you would be mainly interested in the distance between a review and *pwords* and *nwords* along the principal components on which *pwords* and *nwords* lie furthest from each other. The idea would be to stretch the LSA space such that *pwords* and *nwords* would be as far away from each other as possible. This could also be done for other bipolar sets of words.

6.2.3 Parsing

An obvious road to possible improvement is natural language parsing. We discussed Liu & Seneff (2009). Their approach is certainly promising. It was highly involved, however, as parsing is a difficult problem. In analysing some of the mistakes that our classifiers made, we noted that anaphor resolution may be a way to improve performance. This, too, is a very hard and open problem. Parsing enables finding negation in a sentence even if the negation is present in the beginning of the sentence and the clause to which it applies is at the end. As discussed in our related work section, Kennedy & Inkpen (2006) used this to obtain a very small but significant improvement over using unigram features alone.

Whitelaw et al. (2005) also parsed sentences at least partially to find negations and intensifiers. In addition, they used a semi-automatically built lexicon to extract appraisal groups. Using normalized frequency counts of certain types of appraisal groups combined with bag of words features they significantly improved on state of the art performance on the dataset compiled by Pang et al. (2002), lifting it to just over ninety percent. Building the lexicon and parsing the sentences requires much human knowledge of language. It seems linguistics is the most promising way to breach the high ceiling reached with support vector machines on binary unigram features.

Acknowledgements

First of all, I wish to thank my supervisors Dr. Marco Wiering and Dr. Gosse Bouma. Marco was very enthousiastic throughout the project, and offered many interesting ideas. Gosse offered invaluable advice, opening up previously unknown research areas to me with just the right keywords. I also wish to thank my fellow students Jean-Paul, Auke Dirk, Robert, Theije, Jasper and Lise for creating the kind of friendly atmosphere in which it is good to work. Thanks also for Jean-Paul and Mitsai for letting me sleep over when I needed to work at night and for the lovely meals. Last but not least, I wish to my thank my family, my friends and my girlfriend Marloes for supporting me and encouraging me. And many thanks to Marloes and her family for preparing our new home in Amsterdam!

Bibliography

Baroni, M., Chantree, F., Kilgarriff, A., & Sharoff, S. (2008). Cleaneval: a competition for cleaning web pages. In *Proceedings of the Conference on Language Resources and Evaluation (LREC), Marrakech*.

Bradford, R. (2008). An empirical study of required dimensionality for large-scale latent semantic indexing applications. In *Proceeding of the 17th ACM conference on Information and knowledge management*, (pp. 153–162). ACM.

Brand, M. (2006). Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, *415*(1), 20–30.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, *2*(2), 121–167.

Chakraborti, S., Mukras, R., Lothian, R., Wiratunga, N., Watt, S., & Harper, D. (2007). Supervised latent semantic indexing using adaptive sprinkling. In *Proc. of IJCAI*, (pp. 1582–1587).

Church, K., & Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational linguistics*, *16*(1), 22–29.

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

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, *41*(6), 391–407.

Dehling, H., & Kalma, J. (1995). Kansrekening, het zekere van het onzekere. *Epsilon Uitgaven, Utrecht*.

Denhière, G., & Lemaire, B. (2006). Effects of high-order co-occurrences on word semantic similarity. *Current psychology letters. Behaviour, brain & cognition*, (18, Vol. 1, 2006).

Drucker, H., Wu, D., & Vapnik, V. (1999). Support vector machines for spam categorization. *IEEE Transactions on Neural networks*, *10*(5), 1048–1054.

Duda, R., Hart, P., & Stork, D. (2001). *Pattern classification*. Wiley-Interscience.

Duin, R. (1996). A note on comparing classifiers* 1. *Pattern Recognition Letters*, *17*(5), 529–536.

Fletcher, W. (2004). Making the web more useful as a source for linguistic corpora. *Language and Computers*, *52*(1), 191–205.

Herbrich, R., & Graepel, T. (2002). A PAC-Bayesian margin bound for linear classifiers. *IEEE Transactions on Information Theory*, *48*(12), 3140–3150.

Hsu, C., Chang, C., Lin, C., et al. (2003). A practical guide to support vector classification.

Joachims, T. (1999a). SVMLight: Support Vector Machine. *SVM-Light Support Vector Machine http://svmlight. joachims. org/, University of Dortmund*.

Joachims, T. (1999b). Transductive inference for text classifcation using support vector machines. In *International Conference on Machine Learning (ICML)*, vol. 1999.

Joachims, T. (2002). *Learning to classify text using support vector machines*. Springer Netherlands.

Joachims, T., Nedellec, C., & Rouveirol, C. (1998). Text categorization with support vector machines: learning with many relevant features. In *Machine Learning: ECML-98 10th European Conference on Machine Learning, Chemnitz, Germany*, (pp. 137–142). Springer.

John, G., & Langley, P. (1995). Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the eleventh conference on uncertainty in artificial intelligence*, vol. 1, (pp. 338–345). Citeseer.

Kennedy, A., & Inkpen, D. (2006). Sentiment classification of movie reviews using contextual valence shifters. *Computational Intelligence*, *22*(2), 110–125.

Landauer, T., & Dumais, S. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, *104*(2), 211–240.

Landauer, T. K., & Dumais, S. (2008). Latent semantic analysis. *Scholarpedia*, *3*(11), 4356.

Leach, S. (1995). Singular value decomposition-a primer. *Unpublished Manuscript, Department of Computer Science, Brown University, Providence, RI, USA*.

Liu, J., & Seneff, S. (2009). Review Sentiment Scoring via a Parse-and-Paraphrase Paradigm. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, (pp. 161–169). Association for Computational Linguistics.

Liu, L., & Zsu, M. (2009). *Encyclopedia of Database Systems*. Springer Publishing Company, Incorporated.

Manber, U., & Myers, G. (1990). Suffix arrays: a new method for on-line string searches. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, (pp. 319–327). Society for Industrial and Applied Mathematics.

Manning, C., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval.

Manning, C., & Schütze, H. (2000). *Foundations of statistical natural language processing*. MIT Press.

Marsden, J., & Tromba, A. (2003). *Vector calculus*. WH Freeman.

Miranda, A., Le Borgne, Y., & Bontempi, G. (2008). New routes from minimal approximation error to principal components. *Neural Processing Letters*, *27*(3), 197–207.

Mishne, G. (2005). Experiments with mood classification in blog posts. In *Proceedings of ACM SIGIR 2005 Workshop on Stylistic Analysis of Text for Information Access*. Citeseer.

Mishne, G., & de Rijke, M. (2006). Capturing global mood levels using blog posts. In *AAAI 2006 Spring symposium on computational approaches to analysing weblogs*, (pp. 145–152).

Mitchell, T. (1997). *Machine learning. WCB*. Mac Graw Hill.

Pang, B., & Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the ACL*, vol. 2004.

Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, *2*(1-2), 1–135.

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, (pp. 79–86). Association for Computational Linguistics Morristown, NJ, USA.

Sahlgren, M. (2005). An introduction to random indexing. In *Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*. Citeseer.

Shima, K., Todoriki, M., & Suzuki, A. (2004). SVM-based feature selection of latent semantic features. *Pattern Recognition Letters*, *25*(9), 1051–1057.

Stone, P., Dunphy, D., Smith, M., Ogilvie, D., et al. (1966). *The general inquirer: A computer approach to content analysis*. MIT Press Cambridge, MA.

Sun, J., Chen, Z., Zeng, H., Lu, Y., Shi, C., & Ma, W. (2004). Supervised latent semantic indexing for document categorization. In *Fourth IEEE International Conference on Data Mining, 2004. ICDM'04*, (pp. 535–538).

Taboada, M., Anthony, C., & Voll, K. (2006). Methods for creating semantic orientation dictionaries. In *Conference on Language Resources and Evaluation (LREC)*, (pp. 427–432). Citeseer.

Tsur, O., Davidov, D., & Rappoport, A. (2010). ICWSM-A Great Catchy Name: Semi-Supervised Recognition of Sarcastic Sentences in Product Reviews.

Turney, P., & Littman, M. (2003). Measuring praise and criticism: Inference of semantic orientation from association. *ACM Transactions on Information Systems (TOIS)*, *21*(4), 315–346.

Turney, P., et al. (2002). Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, (pp. 417–424).

Vapnik, V. (1982). Estimation of Dependences from Empirical Data.

Vapnik, V. (1998). Statistical Learning Theory. Adaptive and learning systems for signal processing, communications, and control. *Simon Haykin*.

Vapnik, V., & Chervonenkis, A. (1968). Uniform convergence of frequencies of occurence of events to their probabilities. In *Dokl. Akad. Nauk SSSR*, vol. 181, (pp. 915–918).

Whitelaw, C., Garg, N., & Argamon, S. (2005). Using appraisal groups for sentiment analysis. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, (p. 631). ACM.

Yi, L., Liu, B., & Li, X. (2003). Eliminating noisy information in web pages for data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 296–305). ACM New York, NY, USA.

Yu, L., Wang, S., & Lai, K. (2008). Investigation of Diversity Strategies in SVM Ensemble Learning. In *Fourth International Conference on Natural Computation, 2008. ICNC'08*, vol. 7.

Zhu, M., & Ghodsi, A. (2006). Automatic dimensionality selection from the scree plot via the use of profile likelihood. *Computational Statistics & Data Analysis*, *51*(2), 918–930.