Contents

Introduction

The problem of text analysis has been around since the end of the previous century and has since evolved to numerous forms. Other names for this problem are text mining and text analytics. Labor-intensive manual text mining approaches first surfaced in the mid-1980s, but technological advances have enabled the field to advance during the past decade. Text mining usually involves the process of structuring the input text (usually parsing, along with the addition of some derived linguistic features and the removal of others, and subsequent insertion into a database), deriving patterns within the structured data, and finally evaluation and interpretation of the output. Typical text mining tasks include text categorization, text clustering, concept/entity extraction, production of granular taxonomies, sentiment analysis, document summarization, and a plethora of others. The term text analytics describes a set of linguistic, statistical, and machine learning techniques that model and structure the information content of textual sources for business intelligence, exploratory data analysis, research, or investigation.

Sentiment analysis, which refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials, is the most crucial problem for businesses related to social media and market advertisement. An average client may produce about 100 messages per day. What is more, sometimes sentiment data is generated by web crawlers which aggregate data from various sources. Given the amount of data those enterprises poses, it is almost infeasible to process all the messages manually. Therefore, automatic text analytics is indispensable. Even those methods with low precision can generate profits.

A basic task in sentiment analysis is classifying the polarity of a given text at the document, sentence, or feature/aspect level - whether the expressed opinion

in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. Advanced, "beyond polarity" sentiment classification looks, for instance, at emotional states such as "angry," "sad," and "happy." In this work only the basic task is considered.

Existing approaches to sentiment analysis can be grouped into three main categories: knowledge-based techniques, statistical methods, and hybrid approaches. Knowledge-based techniques classify text by affect categories based on the presence of unambiguous affect words such as happy, sad, afraid, and bored. Some knowledge bases not only list obvious affect words, but also assign arbitrary words a probable "affinity" to particular emotions. Statistical methods leverage on elements from machine learning such as latent semantic analysis, support vector machines, "bag of words" and Semantic Orientation - Pointwise Mutual Information.

In this paper I consider using text analysis and statistical methods to perform sentiment analysis of the data refered to specific domain. I leverage different common approaches and suggest improvements to them for solving the particular problem I work on. Moreover, I analyze some of the common problems like feature vector selection. Finally, I provide a performance comparison of the detailed algorithms in practice.

Chapter 1

Statement of the Problem

1.1 Scope and Applications

A marketing campaign is efforts of a company or a third-party marketing company to increase awareness for a particular product or service, or to increase consumer awareness of a business or organization. It has a limited duration. A marketing campaign consists of specific activities designed to promote a product, service or business. A marketing campaign is a coordinated series of steps that can include promotion of a product through different mediums (television, radio, print, online) using a variety of different types of advertisements. The campaign doesn't have to rely solely on advertising, and can also include demonstrations, word of mouth and other interactive techniques.

After a particular campaign is finished, the business is interested in aggregating people's opinion about the products which were being promoted. The vast majority of the reviews come from social media and shoping websites. Basically a review is a short text which is either positive or negative. As was mentioned before, because of large quantities of customers and review sources, manual review analysis is not feasible. The solution is to use an automated review analysis. A classificator can be built on the date that have been already analysed by humans. The larger the corpus the better.

In general, the problem of sentiment analysis depends heavily on the application's domain and can benefit from additional metadata available along with the text data. For example, in case of reviews related to political domain, the analyst might expect some amount of sarcasm and indirect implications. More-

over, a language which is used also plays important role; similar expressions in different languages have different shades of meaning. Therefore, it often is beneficial to target a specific domain and a specific language in text sentiment analysis problem.

In this paper I am going to target customer reviews of baby products in English available on amazon website, the biggest retailer in the world. The data is available at https://s3.amazonaws.com/static.dato.com/files/coursera/course-3/amazon_baby.csv.zip. This data contains 183532 labeled products reviews. The polarity grade is 1-5. 1 - means the most negative review and 5 - the most positive.

1.2 Definitions

Let us denote a set of all possible words W and a set of all possible permutations G. And let R be a set of all the reviews available. So $r \in R$ - product review. $R = \{(w, g) : w \in W, g \in G\}$. P - review polarities set. Each review has corresponding polarity $p = \{1, 2, 3, 4, 5\}, p \in P$. $|R| = |P| = m$.

X feature matrix for machine learning. X is an $m \times n$ matrix, where m is a number of training samples and n is a feature vector cardinality.

Y - label vector for machine learning. In this case it is a mapped polarity of a review. $Y = \{-1, 1\}$, where -1 correspond to negative opinion and 1 - to the positive. $|Y| = m$.

$X_{train} \subseteq X$ - training data.

$X_{test} \subseteq X$ - test data.

$Y_{train} \subseteq Y$ - training labels.

$Y_{test} \subseteq Y$ - test labels.

$extr : R \to X$ - feature extraction function. Given a review r, this functions maps it to the feature vector x.

$pol : P \to Y$ - polarity map function. Maps polarity to binary vector.

$f : X \to Y$ - classification function. Given a feature vector x, this function predicts its label y.

Now text analysis process can be defined. The sets R and P are available. Firstly, an extraction function $extr$ must be defined. After feature extraction process we should be able to form a feature matrix X. Also the representation of the polarities set P is changed by applying a map function pol. Hence, X and Y are defined.

At this moment, I shall split the data into training and cross-validation set. Or in other words training and test data. Now f must be found by a classification algorithm on the training set.

The ultimate goal is to obtain a decision function $F : R \to \{extr, f\}$ that would minimize $\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} (f(x_i) - y_i)^2$ for training data.

TODO precission and recall

Chapter 2

Feature Extraction

2.1 Motivation

Consider the simple case of text classification based on the presence or absence of just one word $w \in W$. Suppose we know that the word w only occurs is reviews with negative opinion and never is present in reviews with positive opinion. This gives us confidence that any message containing w is negative. This approach can be generalized to the probability of a message feature vector occurring in the message.

The entities, I need to classify, are text messages that are given in the form of strings. Raw strings are not convenient objects to handle by classifiers. Most machine learning algorithms can only classify numerical objects or otherwise require a distance metric or other measure of similarity between the objects. Moreover, string by itself doesn't represent a polarity of the message in any way. Therefore, set of features should be extracted.

Before proceeding with machine learning we have to convert all messages to numerical vectors called *feature vectors*, and then classify these vectors. The simplest example of a feature vector is the vector of the numbers of occurrences of all the words in the dictionary in a message. The problem wih this approach is that not all the words affect a sentiment. In particular there are so-called STOP words which are completely useless in this respect. This causes significant performance degradation. Additionaly, this approach does not analyze the context of the word, which causes prediction precission to be lower.

Extraction of features usually means that some information from the original

message is lost. On the other hand, the way feature vector is chosen is crucial for the performance of the filter. There is a trade-off between prediction quality and training speed. Usually, the difference between performance of a really detailed model and a reasonably simple model is not significant. Yet the training speed of a simple model is considerably swifter. Therefore, in most practical applications the most basic vector of word frequencies or its modification is used.

There are plethora of machine learning algorithms which are suitable for review classification task. Choosing the best algorithm may improve a quality of prediction. In practice, however, it is much more important what features are chosen for classification than what classification algorithm is used. If the features are chosen badly, then any machine learning algorithm will fail to classify data correctly no matter how good it is. A reasonable choise of features can make classification both precise and quick.

2.2 Important Words Selection

As already was mentioned in the previous section, selecting all the words as features is not very practical. Yet in performance comparison I will use this approach as well because it can be a baseline for other extraction functions quality. In case of using all the words let us define a word dicionary $D \subseteq W$. Let us suppose $|D| = k|$. Then for matrix X which is $m \times n$ dimensional, now $n = k$. Now $x_{ij} \in X$ represents number of occurances of the word $d_j \in D$ in the review $r_i \in R$.

Now let us consider extracting a smaller dictionary. The problem is to decide which words to include not to lose important features. As it turns out adjectives used most often define an opinion. There is almost no use in adverbs, because of a domain. It is possible that in restaurant reviews there are lots of them, yet for baby products it is not. Also I am going to use link verbs. Common link verbs are: be, appear, look, seem, become, get. For instance, in the sentence "My baby seemed happy". Another reasonable idea is including some verbs such as love, like, enjoy, recommend, etc.

2.3 Context Mining

In the previous section I described in detail the process of words selection. However, before begining classification there is still a work to be done. Consider the following example: "My baby likes this toy", "My baby doesn't like this toy". Using our previous approach there is no difference between those two sentences in classification matrix. But obviously there is a significant difference - pollarity is exactly the opposite. One solution would be to include "not" in the dictionary and hope that classifier will give it significant negative wage. The issue with this approach though is that in more complex reviews, a negation can bear positive sense. Consider an example "I really like this cradle. It looks nice and my baby does not cry as often as before". Therefore, another solution should be found. I suggest using antonyms.

Chapter 3

Classification Techniques

3.1 k Nearest Neighbors Classifier

k Nearest Neighbors (KNN) is part of supervised learning that has been used in many applications in the field of data mining, statistical pattern recognition, image processing and many others. The idea is to predict class for a given feature vector using so called majority voting. Suppose we want to predict label for a feature vector $x_i \in X_{test}$. Then we look on labels of k training vectors which are "the closest" to x_i and use the most frequent label as a prediction. To define "closeness" some distance metric is used. There are several options for distance functions.

A commonly used distance metric for continuous and discrete variables is Euclidean distance:

$$\sum_{i=1}^{n}(a_i - b_i)^2$$

, where a,b - feature vectors. $a_i - b_i$ - vector difference.

For discrete variables, such as for text search, another metric can be used, such as the overlap metric (or Hamming distance).

The best choice of k depends upon the data; generally, larger values of k reduce the effect of noise on the classification, but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques. In our case though I am going to use a fixed value.

This alorithm does not require training step. Having sets X_{train} and Y_{train} and distance function $dist$ is enough for making predictions. On the other hand, the

classification of test data takes time, because for every review without label, the whole training corpus must be processed to calculate distance. For one prediction expected time complexity is $O(m_{train} * n)$. Performing indexing on the training set can decrease the running closer to $O(n)$. However, indexing is a computationally heavy operation and in case data is updated often, this approach becomes infeasible. Moreover, precission of this classification algorithm is usually low compared to others. Another drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number.

In practice KNN algorithm performs worse than other algorithms. Therefore KNN model is mostly used as a baseline to compare it with other models.

3.2 Naive Bayes Classifier

Naive Bayes algorithm is a classification algorithm based on Bayes' theorems, and can be used for both exploratory and predictive modeling. The word naive in the name Naive Bayes derives from the fact that the algorithm uses Bayesian techniques but does not take into account dependencies that may exist. This algorithm is less computationally intense than most other classification algorithms, and therefore is useful for quickly generating mining models to discover relationships between input columns and predictable columns.

Naive Bayes relies on very simple representation of the document - Bag of words.

Input:

- A document d
- A fixed set of classes $C = \{c1, c2, ..., c_j\}$
- A training set of m hand-labeled documents $(d1, c1), (d2, c2), ..., (d_m, c_m)$

Output:

- A learned classifier $y : d \to c$.

For a document d and class c

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

.

This probability can be used to predict a class of a document.

$$c_{best} = \underset{c \in C}{\operatorname{argmax}} P(c|d) = \underset{c \in C}{\operatorname{argmax}} \frac{P(d|c)P(c)}{P(d)} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

Taking into account that d is a bag of words:

$$c_{best} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c) = \underset{c \in C}{\operatorname{argmax}} P(w_1, w_2, ..., w_n|c)P(c)$$

Now let us find the way to compute those two probailities. Let us start with $P(c)$. We can answer the question how often this class occurs. It can be computed by counting relative frequencies in a corpus.

Computing $P(w_1, w_2, ..., w_n|c)$ is more subtle. There is $|W|^n * |C|$ parameters to be computed, which is impossible in practice. In Naive Bayes classifier two simplifying assumptions are made:

- **Bag of Words assumption:** Assume position does not matter
- **Conditional Independence:** Assume the feature probabilities $P(x_i|c)$ are independent given a class c.

Despite these simplification in practice a problem can be solved with a high degree of accuracy. The result of the simplyfying assumptions is the following equation:

$$P(w_1, w_2, ..., w_n|c) = P(w_1|c) \times P(w_2|c) \times ... \times P(w_n|c)$$

Therefore:

$$c_{best} = \underset{c \in C}{\operatorname{argmax}} P(w_1, w_2, ..., w_n|c)P(c) = \underset{c \in C}{\operatorname{argmax}} P(c) \prod_{w \in W} P(w|c)$$

Now we have to find the way to compute these probabilities. The most natural way to implement this is simply to use frequencies in the data.

$$P(w_i|c) = \frac{doccount(C = c)}{N_{doc}}$$

$$P(c) = \frac{count(w_i, c)}{\sum_{w \in W} count(w, c)}$$

So I am going to compute the fraction of times word w_i appears among all words in documents of class c. To do that it is possible to concatenate all available documents of a particular class c into one mega-document. In case of this paper there is two mega-docuemtns D_{pos} and D_{neg}, which correspond to positive and negative reviews.

Unfortunatelly, there is a problem with the approach described above. What should happen if we have seen no training example with the word **fantastic** and classified in the positive?

$$P(\text{"}fantastic\text{"}|positive) = \frac{count(\text{"}fantastic\text{"}, positive)}{\sum_{w \in W} count(w, positive)} = 0$$

As can be seen, even though the word **fantastic** is usually associated with positive sentiment, the probability of a document which contains this word belongs to D_{pos} is zero. Here is why:

$$c_{best} = \underset{c \in C}{\operatorname{argmax}} P(c) P(\text{"}fantastic\text{"}|c) \prod_{w \in W / \{\text{"}fantastic\text{"}\}} P(w|c)$$

For positive class this product is equal to zero.

The solution to this problem is adding Laplace (add-1) smoothing for Naive Bayes. Simply one is added to each of the counts:

$$P(w_i|c) = \frac{count(w_i, c) + 1}{(\sum_{w \in W} (count(w, c) + 1)} = \frac{count(w_i, c) + 1}{\sum_{w \in W} count(w, c)) + |W|}$$

TODO

Consider the simple case of text classification based on the presence or absence of just one word W. Suppose we know that the word W only occurs is spam messages. This gives us confidence that any message containing W is spam. This approach can be generalized to the probability of a message feature vector occurring in the message.

Suppose we have two classes L and S corresponding to legitimate and spam messages, and that there is a known probability distribution of feature vectors $P(x|c), c \in \{L, S\}$. In general it is hard to define such distribution, but it is often possible to provide an approximation. What we need to obtain is the class that the given message belongs to, or the probability $P(c|x)$. This can be done using the Bayes' formula

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} = \frac{P(x|c)P(c)}{P(x|L)P(L) + P(x|S)P(S)}$$

.

where $P(x)$ is the a-priori probability of a message with feature vector x and $P(c)$ is the probability of class c, i.e. the probability that any given message belongs to c. Given the values $P(c)$ and $P(x|c)$ for $c \in \{L, S\}$ one can calculate the probability $P(c|x)$ which can then be used in a classification rule.

The most basic classification rule is to classify message to the category with bigger probability.

Definition 3.2.1. *Maximum a-posteriori probability (MAP) rule: if $P(S|x) > P(L|x)$ then classify x as spam, otherwise classify as legitimate message.*

The MAP rule can be transformed to

If $\dfrac{P(x|S)}{P(x|L)} > \dfrac{P(L)}{P(S)}$ then classify x as spam, otherwise as legitimate message.

The ratio $\dfrac{P(x|S)}{P(x|L)}$ is known as the *likelihood ratio* and is denoted as $\Lambda(x)$.

This approach can be too simplistic for certain applications. For example, in case of e-mail spam filtering, false positives (classifying legitimate message as spam) are usually much more unwanted than false negatives (classifying spam as legitimate message). The following generalization allows to take such restrictions into account.

Definition 3.2.2. *A cost function $\mathcal{L}(c_1, c_2)$ denotes the cost of misclassifying a message of class c_1 as the one belonging to class c_2.*

It is natural to put $\mathcal{L}(L, L) = \mathcal{L}(S, S) = 0$, but in general it might not be the case.

Then we can express the expected risk of classifying a message x belonging to class c in the above terms.

Definition 3.2.3. *The function $R(c|x) = \mathcal{L}(S, c)P(S|x) + \mathcal{L}(L, c)P(L|x), x \in M, c \in \{L, S\}$ is called the risk function.*

Now we can define a natural classification rule in terms of expected risk.

Definition 3.2.4. *Bayes' classification rule: if $R(S|x) < R(L|x)$ then classify x as spam, otherwise as legitimate message [?].*

It can be shown that Bayesian classifier f minimizes the average risk

$$R(f) = \int \mathcal{L}(c, f(x))dP(c, x) = P(L) \int \mathcal{L}(L, f(x))dP(x|L) + P(S) \int \mathcal{L}(S, f(x))dP(x|S)$$

so in this sense Bayesian classifier already is optimal [**?**].

Naturally, the loss of classifying the message correctly is zero, thus $L(S, S) = L(L, L) = 0$. Then the Bayes's classification rule can be rewritten as

$$\text{If } \Lambda(x) > \lambda \frac{P(L)}{P(S)} \text{ classify as spam otherwise as legitimate message.}$$

Here $\lambda = \dfrac{\mathcal{L}(L, S)}{\mathcal{L}(S, L)}$ is the additional parameter that specifies the risk of misclassifying legitimate messages as spam. As the value of λ increases, the classifier produces fewer false positives.

While the classification process is straightforward, the practical applications of Bayes's classifier are limited by our ability to approximate the a-priori probabilities $P(x|c)$ and $P(c), c \in \{L, S\}$ from the training data. Therefore, while the Bayes's classifier is optimal in the sense of minimizing the loss of classification for given a-priori probabilities, the quality of spam detection depends on the feature selection and approximation of these probabilities.

$P(L)$ and $P(S)$ can be easily approximated by the ratio of legitimate and spam messages respectively. $P(x|c)$ is non-trivial and depends on the contents of selected feature vector. Consider the simplest case where the feature vector x_w is 1 if the message contains w and 0 otherwise. Then the probability $P(x_w = 1|S)$ can be approximated by the ratio of spam messages containing w to the ratio of all spam messages in a training set. This is sufficient to be used by the Bayes's classifier, so we can outline the training and selection process for this model.

Training process

1. Calculate probabilities $P(c)$, $P(x_w = 0|c), P(x_w = 1|c), c \in \{L, S\}$.
2. Using Bayes's formula calculate $P(c|x_w = 0)$ and $P(c|x_w = 1)$.
3. Calculate $\Lambda(x_w), x_w = 0, 1$, calculate $\lambda \dfrac{P(L)}{P(S)}$ and store these values.

Classification process

1. Determine feature vector x_w for message m.
2. Retrieve the stored value $\Lambda(x_w)$.
3. Use Bayes's decision rule to determine class of the message.

Now we need to generalize this classifier to include more features than just the presence of a single word. The simplest way (and a very common one) is to choose a set of most common words $w_1, w_2, \ldots, w_n$ and define the feature vector $x = (x_1, x_2, \ldots, x_n), x_i =, x_i = 1$ if the message contains w_i, $x_i = 0$ otherwise.

The problem with this approach is that it requires calculation and storing of all possible values of the feature vector, and there are 2^n such vectors, which is not feasible. A common way to remove this requirement is to assume that the individual components of the vector are independent [**?**]. This assumption is not formally correct, but in practice it is a good compromise between formal correctness and computational requirements. We will consider other options in later chapters.

Because of independence of features:

$$P(x|c) = \prod_{i=1}^{n} P(x_i|c)$$

$$\Lambda(x) = \prod_{i=1}^{n} \Lambda_i(x_i)$$

This classifier is known as Naive Bayesian Classifier due to assumption of independence of features. Training and classification are very simple computationally.

Training process

1. For all $w_i \in W$ calculate and store $\Lambda_i(x_i), x = 0, 1$.
2. Calculate and store $\lambda \dfrac{P(L)}{P(S)}$.

Classification process

1. Determine feature vector x for message m.
2. calculate $\Lambda(x)$ using the stored values $\Lambda_i(x_i)$.
3. Use Naive Bayes's decision rule to determine class of the message.

In terms of word selection for the feature vector, usually words that are too common or too rare are excluded. For simple cases when performance is not critical, all words form the training set can be used. In later chapters we will consider ways to select words with maximum mutual information.

Another benefit of naive Bayesian filter is that it is very easy to expand the feature vector to include additional available metadata. In case of e-mails, for example, it would be contents of e-mail headers. It is possible to include additional components either to the calculation of the a-priory probability of the vector or to combine the risk of Bayesian classifier with additional risk calculated from metadata when making a decision.

3.3 SVM Classifier

Support Vector Machines (SVM) is a class of widely used algorithms for classification and regression developed by V. Vapnik. The theoretical foundation of SVM is the Statistical Learning Theory that gives certain guarantees of performance. Let us consider classification problem with SVM for linearly separable data.

SVM works in a similar manner to perceptron in terms of finding a linear boundary that separates test data according to their classes. However, the purpose of SVM is not to find any of these boundaries if they exist, but to find the maximal margin separating hyperplane, for which the distance to the closest training sample is maximal.

Definition 3.3.1. *Let* $X = \{(x_i, c_i)\}$, $x_i \in \mathbb{R}^n$, $c_i \in \{-1, +1\}$ *be the set of training samples. Suppose* (w, b) *is a separating hyperplane* $sign(w^T x_i + b) = c_i$ *for all* $1 \leq i \leq k$. *The margin* m_i *of a training sample* (x_i, c_i) *with respect to the separating hyperplane is the distance from* x_i *to the hyperplane*

$$m_i = \frac{|w^T x_i + b|}{||w||}$$

.

The margin m *of the separating hyperplane for training set* X *is the smallest margin of an instance in the training set*

$$m = \min_i m_i$$

.

The maximal margin separating hyperplane for training set X *is the separating hyperplane with maximal margin with respect to the training set [?].*

Because the hyperplane given by parameters (x, b) is the same as the hyperplane given by parameters (kx, kb), we can safely bound our search by only considering canonical hyperplanes for which $\min_i |w^T x_i + b| = 1$. It is possible to show that the optimal canonical hyperplane has minimal $||w||$, and that in order to find a canonical hyperplane it suffices to solve the minimization problem: minimize $\frac{1}{2} w^T w$ under conditions

$$c_i(w^T x_i + b) \geq 1, i = 1, 2, \ldots, k$$

The problem may be transformed to a certain dual form: maximize

$$L_d(\alpha) = \sum_{i=1}^{k} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{k} \alpha_i \alpha_j c_i c_j x_i^T x_j$$

with respect to dual variables $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k)$, $\alpha_i \geq 0 \forall i$ and $\sum_{i=1}^{k} \alpha_i c_i = 0$.

This is a classical quadratic optimization problem. It mostly has a guaranteed unique solution, and there are efficient algorithms for finding this solution. Once we have found the solution α, the parameters (w_o, b_o) of the optimal hyperplane are determined as

$$w_o = \sum_{i=1}^{k} \alpha_i c_i x_i$$

,

$$b_o = \frac{1}{c_m} - w_o^T x_k$$

where m is an arbitrary index for which $\alpha_m \neq 0$.

It is more-or-less clear that the resulting hyperplane is completely defined by the training samples that are at minimal distance to it. These training samples are called support vectors and thus give the name to the method. It is possible to tune the amount of false positives produced by an SVM classifier, by using the so-called soft margin hyperplane and there are also lots of other modifications related to SVM learning. Recall the training and classification process.

Training process

1. Find α that solves the dual problem (maximizes L_d under named constraints).
2. Determine w and b for the optimal hyperplane and store the values.

Classification process

1. For message with feature vector x classify it as $sign(w^T x + b)$.

3.4 Decision trees

bla bla bla

3.5 Boosting

AdaBoost.

3.6 Artificial Neural Network Classifier

Artificial neural networks (ANN) is a family of models inspired by biological neural networks which are widely used in classification, regression and density estimation by approximating functions that can depend on a large number of inputs and are generally unknown. A neural network is a complex function that may be decomposed into smaller units called neurons and represented graphically as a network. Many functions fall under such criteria, however the most common kinds of neurons are perceptron and multilayer perceptron.

The perceptron produces a linear function of the feature vector $f(x) = w^T x + b$ where $f(x) > 0$ for vectors of one class and $f(x) < 0$ for vectors of another class. Here w is the vector of weights, or *bias*, $w = (w_1, w_2, \ldots, w_n)$. This vector will be determined by the training process.

If we denote the classes by number -1 and +1, we can use $d(x) = sign(w^T x + b)$ as decision function. This allows us to represent the decision function graphically as a neuron with n inputs and a single output. A system of one perceptron is an example of the simplest neural network.

Suppose the feature vector is two-dimensional, $x \in \mathbb{R}^2$. Then we can represent these feature vectors as points on the plane. Then the decision function can be represented as a line dividing the plane in two parts, each corresponding to one of the classes. Similarly, the decision boundary for three-dimensional feature vectors is a plane, etc. In general, for n-dimensional feature vector the decision boundary is a n - dimensional hyperplane.

The learning process for a perceptron is iterative. The initial values of parameters (w_0, b_0) can be arbitrary, as they are updated on each iteration. On the k-th iteration of the algorithm a training sample (x, c) is chosen such that the current decision function does not classify it correctly, i.e. $sign(w_k^T x + b_k) \neq c$. Then the parameters (w_k, b_k) are then updated according to the rule:

$$w_{k+1} = w_k + cx$$

,

$$b_{k+1} = b_k + c$$

.

The algorithm terminates when a decision function that correctly classifies all training set is found. If the training set is linearly separable, the perceptron algorithm converges. It is known as Perceptron Convergence Theorem proven by Frank Rosenblatt in 1962 [**?**]. If, however, the set is not linearly separable, the algorithm will never converge. In this case it is possible to still use the perceptron, but the training loop needs to stop when the number of misclassification becomes small.

We can now outline the training and classification phases of the perceptron.

Training process

1. Initialize the values of w and b with random values or 0.
2. Find a sample from the training set (x, c) such that $sign(w^T x + b) \neq c$. If there are no such samples, terminate as the training is completed and all training samples are being classified correctly, else proceed to the next step.
3. Update (w, b) with new values $w := w + cx$, $b := b + c$ and go to the previous step.

Classification process

1. For message with feature vector x classify it as $sign(w^T x + b)$.

As mentioned before, perceptrons can be combined in multiple layers to form *multilayer perceptrons* which are non-linear classifiers. Neurons of the first layer which takes in the input parameters are called *input neurons*, similarly neutrons of the last layer which provide the function result value are called *output neurons*. All layers between input and output are called *hidden layers*.

Each neuron in the networks is similar to a perceptron: for input vector $x = (x_1, x_2, \ldots, x_n)$ it calculates output value by the formula

$$o = \phi(\sum_{i=1}^{n} w_i x_i + b)$$

where w_i and b are weights and bias of the neuron respectively, ϕ is a nonlinear function that approximates binary output of the perceptron. Most often $\frac{1}{1+e^{ax}}$ or

$\tanh(x)$ are used as ϕ as they tend to give a good approximation while being mathematically convenient.

Like in the case of a single perceptron, training of a neural network is searching for the values of weights and biases for all neurons that minimize the output error. Let us denote $f(x)$ as the output of the neural network. Then for training samples $(x_i, c_i), 1 \leq i \leq k$ the training has to minimize the *total training error*

$$E(f) = \sum_{i=1}^{k} |f(x_i) - c_i|^2$$

.

An iterative algorithm can be used to perform this minimization. The most common one is the gradient descent which in case of neural networks is called *error backpropagation*. The theory of backwards propagation of errors is well developed and has many implementations in practice [**?**].

The main reason to use multiple layers of neurons is that the multilayer neural network is a non-linear classifier. As a result, they are applicable for tasks with training data that is not linearly separable, particularly when the number of features is relatively small. However, in case of spam detection with multiple words being used as features the data is often linearly separable, thus using neural network will have no noticeable benefits over a simple perceptron.

Performance of the neural network is proportional to the number of neurons. Thus, the large number of features directly impacts performance as it translates to increased number of input neurons and thus the complexity of the network in total. In practice the number of features would have to be more strictly limited than in case of a perceptron, which for spam detection means the trade-off between non-linear decision boundaries and the amount of information loss.

Because of the above reasons and due to the large number of parameters that require tuning. the multilayer perceptron is hard to use in practice for spam detection. It has been successfully used for that purpose [**?**], but it is not easily applicable in general case as it is hard to reconfigure. For the purposes of this paper we shall focus on a simple perceptron.

Chapter 4

Performance Comparison

Words, tables, graphs, pictures, code.

Conclusion

Words.

References

[1] Konstantin Tretyakov. Machine Learning Techniques in Spam Filtering. Institute of Computer Science, University of Tartu. Data Mining Problem-oriented Seminar, MTAT.03.177, May 2004, pp. 60-79.

[2] V. Kecman. Learning and Soft Computing. 2001, The MIT Press.

[3] S. Haykin. Neural Networks: A Comprehensive Foundation. 1998, Prentice Hall.

[4] N. Cristianini, J. Shawe-Taylor. An Introduction to Support Vector Machines and other kernel-based learning methods. 2003, Cambridge University Press. http://www.support-vector.net

[5] Xavier Carreras, Lluis Marquez. Boosting Trees for AntiSpam Email Filtering. TALP Research Center, LSI Department, Universitat Politecnica de Catalunya.