Using a Bayesian Neural Network as a Tool for Document Filtering Considering User Profiles

MAGNUS ERICMATS

KTH Computer Science
and Communication

Master of Science Thesis
Stockholm, Sweden 2013

Using a Bayesian Neural Network as a Tool for Document Filtering Considering User Profiles

MAGNUS ERICMATS

2D1021, Master's Thesis in Computer Science (30 ECTS credits)
Degree Progr. in Engineering Physics 270 credits
Royal Institute of Technology year 2013
Supervisor at CSC was Anders Lansner
Examiner was Anders Lansner

TRITA-CSC-E 2013:016
ISRN-KTH/CSC/E--13/016--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

This thesis describes methods and problems when using Bayesian Artificial Neural Networks for text document classification. It also depict other methods used in text analysis and automated classification in general. The main tasks are to construct a network, investigate the effect of variations to existing parameters and how to combine dependent input attributes into complex columns. Correlation measures are used to find these combinations. The basic idea is to let the classifier built work as a document filtering system. Results from the testing are described and explained.

The results are discouraging. All tests indicate that the training set is too small. Compared to another study done, on the same data, at Swedish Institute of Computer Science the performance of the classifier is poor.

Referat

Nyttjande av ett Baysianskt Neuronät för dokumentfiltrering med hänsyn till användarprofiler

Den här rapporten beskriver metoder och problem vid användande av Bayesianska artificiella neuronnät för dokumentklassificering. Det berör även andra metoder som används inom textanalys och automatisk klassificering. Den huvudsakliga uppgiften är att undersöka effekten av parametrar och variation av dessa och hur beroende indata attribut skall kombineras till att skapa komplexa kolumner. För att hitta dessa kombinationer används korrelationsmått. Grundtanken är att låta den skapade klassificeraren fungera som ett dokumentfiltreringssystem. Resultat från tester är beskrivna och förklarade.

Resultaten är nedslående. Alla tester tyder på att träningsmängden är för liten. Jämfört med en annan studie genomförd, på samma data, vid Swedish Institute of Computer Science så är klassificerarens prestanda låg.

Preface / Acknowledgements

This thesis is a report of a Masters thesis project in computer science at the Royal Institute of Technology (KTH). The work was mainly performed at the Swedish Institute of Computer Science (SICS), during the spring 2000. PhD Anders Holst was the supervisor at SICS, and Professor Anders Lansner was supervisor, examiner and head of the SANS research group at the Royal Institute of Technology.

The project was performed within the ARC, Adaptive Robust Computing laboratory at SICS. Gunnar Sjödin is the head of ARC which is a laboratory that "aims at better understanding the mechanisms for interaction used by natural and artificial intelligent systems, such as humans, animals, robots and other autonomous intelligent agents".

I would like to take the opportunity to express my deepest gratitude to my supervisor Anders Holst for his great suggestions and advice. Without his guidance I would have not be able to complete this project. He also let me use his PhD thesis as a base for this project. Chapter 3 is based on the theoretics described in his PhD thesis [Holst, 1997], which is the best written litterature on the subject.

Further my thanks go to Anders Lansner who lead me into the interesting path towards the world of Artificial Neural Networks and who introduced me to the task.

Daniel Gillblad for the additional understanding of the Bayesian Artificial Neural Network. When Holst wasn't in his office I ran to Gillblad for help.

Douglas Wikström and Fredrik Fyring for sharing the office room with me and making the days go faster.

Annika Waern for the input from her point of view including all information about text analyzing basics. She also provided me with the database used in my tests.

All members of ARC for making the working atmosphere comfortable and for the numerous fun lunch discussions.

Contents

Chapter 1

Introduction

There is a vast amount of written information that becomes available to people every day and some sort of automated classification of text data has become extremely required.

There are many kinds of methods that can be used to perform classification tasks and they all are suitable for different domains. It is often the problem that forms the specific method, but there are methods that designed to be applicable for many sorts of problems.

To categorize text you are forced to use techniques from different science areas, including text parsing, information retrieval and data classifying algorithms.

In this thesis the Bayesian Neural Network is used along with classic linguistic analysis methods, in a hope of creating a good document classifier, such as a spam-filter.

1.1 Overview of the thesis

Chapter 2 constitutes the obligatory literature search. It handles methods used for text analysis. First, the basics of Information Retrieval are described, including measures used in the field. The chapter also covers classification methods in general, including the Artificial Neural Network, the Bayesian Artificial Neural Network, the Augmented Bayesian classifier, Latent Semantic Indexing, Hierarchical Indexing and Decision Trees. Five applications which are used for document classification are described.

Chapter 3 describes the Bayesian Artificial Neural Network in detail. It begins with a brief description of the feed-forward Artificial Neural Network. Then the

Naive Bayesian classifier is described. Further on the classifier is extended with the Bayesian learning rule to form a one-layer Bayesian Artificial Neural Network. At the end the multi-layer Bayesian Artificial Neural Network is described with its hidden layers, including the partitioning, overlapped and fragmented complex columns.

In Chapter 4 the Bayesian Artificial Neural Network is studied when it is used as a document classifier, working in an text environment of conference calls. The idea is to build user profiles corresponding to interests of a set of test subjects. With the user profile you may predict if a 'new' document is relevant or irrelevant to the subject and thus have the possibility to filter it out.

The results are compared to the results of a survey based on the same text collection.

Finally, chapter 5 discusses the results and their reasons. It describes further variations to the approach used. Sources of error are discussed and at the end you find some thoughts of the writer.

Chapter 2

Methods used in text analysis

In this section we mention some of the most frequently used methods that are related and relevant to text classification/categorization; Information Retrieval and algorithms for data classifying.

This chapter represents the literature search for the Thesis Project.

2.1 Information Retrieval

The goal of an Information Retrieval (IR) system is to find relevant documents of some topic of interest. However, ’relevance’ is a vague term. How should ’relevance’ be defined? As [Mizzaro, 1996] writes, there are several kinds of relevance, such as ’utility’, ’usefulness’, ’topicality’ and many more. This is probably the reason why it is so complicated to reach good effectiveness of the IR systems. Many techniques have been developed to explore the meaning of ’relevance’. The following sections discuss some of these measures.

2.1.1 Term frequency - Inverse Document frequency

Term frequency - Inverse Document frequency, $tfidf$, is a measure of how frequent a word (or term) is in a document related to how frequent it is overall; in other words, how significant the term is, [Salton and Buckley, 1988].

If the term is highly frequent in a document it is probably, in some feature meaning, important for that document. But if the term is highly frequent in all document it is not that important, i.e. as the term ’the’. Hence, we are interested in the inversed document frequency.

This measure uses the term frequency, tf_{if} - the number of occurences of term T_i in document D_j and the document frequency, df_i - the number of documents that contain the term T_i.

With these two measures we can write the $tfidf$ measure as:

$$w_{ij} = tf_{ij} * \log_2 \left(\frac{N}{df_i} \right) \tag{2.1}$$

where N is the total number of documents.

This measure normalizes the term occurrences and gets us a nice representation of the document set.

2.1.2 Precision and Recall

In Information Retrieval there are two measures which are frequently used, named Precision and Recall, [EAGLES, 1995]. These measures focuses on the relevant behavior of a system, which in a document retrieval system is to retrieve interesting documents. The precision p measures how many of the retrieved documents ret are relevant $relret$, and the recall r measures how many of the existing relevant documents $reldat$ are actually retrieved $relret$.

$$p = \frac{relret}{ret} \tag{2.2}$$

$$r = \frac{relret}{reldat} \tag{2.3}$$

You must have the documents predefined by the user, to be able to use these measures.

2.1.3 Mutual Information

When you want to measure how much information is achieved when given some data, you must look at the field of Information Theory. The Information Theory discuss the efficiency of information representation, and limitations involved in the reliable transmission of information.

One important concept in Information Theory is the *entropy* measure. Entropy is a measure of order, and is borrowed from the thermodynamics. In Information

Theory it is a measure of the *average amount of information conveyed per message.*

$$H(X) = -\sum_{i} P(x_i) \log P(x_i) \tag{2.4}$$

where $-\log P(x_i)$ is the amount of information we get if we are told that x_i, with probability to occur, $P(x_i)$, has occurred.

Another concept from this domain is the Mutual Information. In a classifier the objective is to learn a input-output mapping. Here, mutual information is of importance. The mutual information is a measure of how much two objects X and Y have in common, and it is based on the definition of the *conditional entropy* [Haykin, 1998]:

$$H(X|Y) = H(X,Y) - H(Y)$$

where $H(X,Y)$ is the joint entropy. This represents the *amount of uncertainty remaining about the input X after the output Y has been observed.*

Since $H(X)$ represents the uncertainty about the system input *before* observing the system output and $H(X|Y)$ represents the uncertainty *after* observing the system output, the difference

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) && (2.5) \\ &= H(X) + H(Y) - H(X,Y) && (2.6) \\ &= \sum_{i,j} P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i)P(y_j)} && (2.7) \end{aligned}$$

must represent the uncertainty about the system input resolved in observation of the output. This is called the *Mutual Information* between variables X and Y.

Note that the Mutual Information is symmetric, $I(X;Y) = I(Y;X)$, and always non negative.

Maximum Mutual Information

When using a measure, one may want to get some understanding in how good/bad a specific measurement is. One way to get this is to compare it to the maximum of the measure. To find the maximum of the Mutual Information for one object relation one may like this:

Set

$$P(y|x) = 1$$

and

$$P(y) = P(x)$$

The first equality means that you are sure in the prediction of Y given X. You may also set the conditional probability to zero, and be sure of the prediction. The two equalities together say that an input attribute is always occurring in a certain class. Now you'll get the maximum mutual information between X and Y, $I(X;Y)_{max}$

$$I(X;Y)_{max} = \sum_{i,j} P(x_i) \log \left(\frac{1}{P(x_j)} \right) \tag{2.8}$$

$$= -\sum_{i} P(x_i) \log P(x_i) \tag{2.9}$$

$$= -P(x_i) \log P(x_i) - (1 - P(x_i)) \log (1 - P(x_i)) \tag{2.10}$$

It will be zero at the extreme points $P(X) = 0$ and $P(X) = 1$. This is rather obvious, because the stochastic variable X contains no information at these points. The highest value of the measure we get at $P(X) = 0.5$. Using base-2 logarithms and plotting the Maximum mutual information versus $P(X)$ we will get a parable through these points, see figure (2.1).

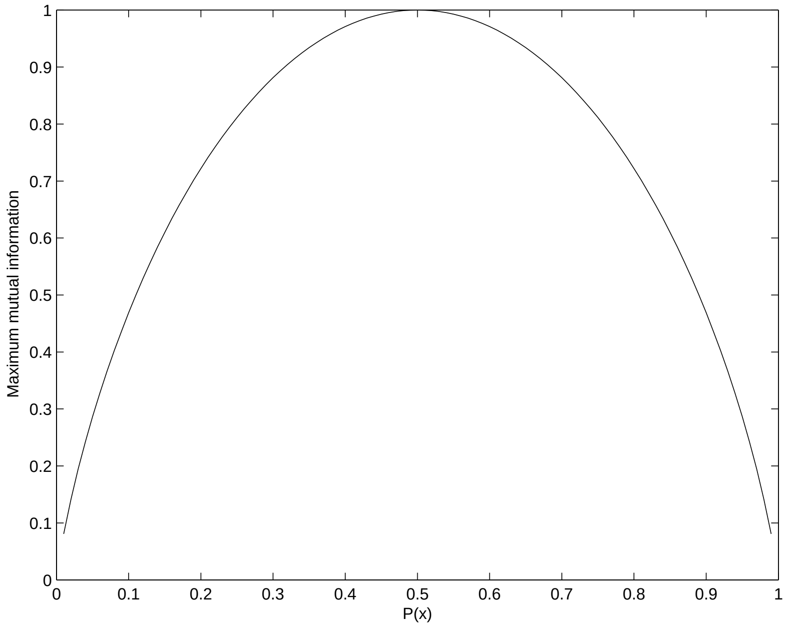

Figure 2.1. Plot showing the maximum mutual information versus $P(X)$.

2.2 Classification

There are several algorithms used for classification, including singular value decomposition, statistical methods, genetic algorithms and artificial neural networks. The purpose with a classification system is to find general features in the dataset and make a generalized classification with them.

Text classification/categorization is the process of algorithmically analyzing an electronic text, and it can be used for filtering purposes.

This section will describe a few methods used for classification; the Artificial Neural Network, the Bayesian Artificial Neural Network, the Augmented Bayesian classifier, Latent Semantic Indexing, Hierarchical Indexing and Decision trees.

Hierarchical indexing and Decision trees are very similar in the way of splitting the categorization problem into smaller problems while Artificial Neural Networks and Bayesian Artificial Neural Networks have topological similarities.

2.2.1 The Artificial Neural Network

The concept of Artificial Neural Networks (ANN) has been motivated by recognition of the computational flow in the human brain [Haykin, 1998], and it tries to resemble the architectural solution of the brain.

The brain is a very complicated computational processor. Apart from the conventional computer, with one big fast computation unit, the human brain consists of about 10^{11}, in the circumstance slow, computation units called *neurons*.

The main function of the neuron is to filter input signals, in form of electric impulses. A neuron is built up by several input branches connected to a cell body. One output branch is also attached to the cell body.

The signals flows from the input branches, through the cell body and sometimes out into the output branch, and there it is transmitted to other neurons. This forwarding will only occur if the sum of input signals to the neuron is strong enough.

The base element of the ANN is the artificial neuron, which in many aspects simulates the functions of the biological neuron. It has properties like nonlinearity, input-output mapping, and fault tolerance.

One further important feature of the neuron is the learning capability. It is done with dynamic input branches, which can be week or strong in their capability of transmitting the signals. In the ANN that is interpreted as weights, and by changing the weights of the neuron you can change the output response of input signals; the ANN 'learns'.

All the above properties of the neuron make it very dynamic and suitable as a part of a big cluster or *network*. In the ANN the neurons are coupled to each otherr as the neurons in the brain, but in a simplified manner. The way the neurons are coupled is called an *architecture*. There exist several architectures, but the most commonly used is the *feed forward* architecture, where no signal loops exists, as in a *recurrent network*. In a recurrent neural network the output of one calculation step is used in the next step of calculation. This is repeated until the network stabilizes.

In the feed forward network, the neurons are placed in layers. Neurons in one layer get their input from the 'previous' layer, and forward their outputs to the 'next' layer. The commonly used *Multi-layer Perceptron* is such a layered network.

The learning is handled by a *learning algorithm*. The most popular type of algorithms is the *Backward Error Propagation* algorithms, usually called *Back-Prop*. Here, the difference between the output and an expected output is propagated backwards through the network layers, in order to change the weights. The change is made to minimize the *error signal* of each neuron in a Least Square Error manner.

The ANN can be seen as a generalized associative memory, as it couples one generalized input to one generalized output. The fact that the ANN can and will generalize is the main reason for using it for classification tasks.

2.2.2 The Bayesian Artificial Neural Network

The Bayesian Artificial Neural Network, is an extension of the Bayesian classifier and was originally built as a recurrent one-layer network, by Lansner &Ekeberg at the SANS-group at the Royal Institute of Technology. The Bayesian Neural Network is trained according to the Bayesian learning-rule, where the neurons in the network represent stochastic events and the weights are calculated based on correlations between them.

Topologically, the Bayesian Neural Network used in this thesis resembles the architecture of the feed-forward Artificial Neural Network.

The one-layer Bayesian Neural Network is built on the Bayesian classifier which assumes independence between the input attributes. That assumption is not always correct. That problem is solved in the multi-layer Bayesian Neural Network by introducing hidden columns representing combinations of input attributes.

The Bayesian Artificial Neural Network will be described in detail in Chapter (3).

2.2.3 Augmented Bayesian classifier

Another way of taking care of the independence assumption is the Augmented Bayesian classifier. The augmented Bayesian classifier augments the original Bayes classifier with correlation arcs between the attributes [Keogh and Pazzani, 1999]. The attributes become independent given the class. You do not want to find the underlying probability distribution, but are more interested in finding a representation that improves the classification accuracy, [Keogh and Pazzani, 1999].

The goal is to calculate the probablity of an instance belonging to class y, $P(y|\overline{x})$. Initially this is equal to the bayesian classifier, but we strive to augment it to improve the result.

The augmented naive Bayesian classifier is defined by the following conditions:

- Each attribute has the class attribute as a parent.
- Attributes may have one other attribute as a parent.

A node without a parent, other than the class, is called an *orphan*.

The second condition results in a dependency arc between the two attribute nodes. When a depency arc from node x_1 and x_2 is formed the above probablity is adjusted by multiplying by $P(x_2|y, x_2)/P(x_2|y)$.

To find suiting additional arcs (between the nodes) one have to use a search algorithm, Keogh and Pazzani make use of a hill climbing greedy algorithm, where they iterative add arcs which best improve the performance till no significant improvement is made.

They also make use of a more efficient search, SuperParent, which gets the same accuracy with less work.

The additional arcs mitigate the independence assumption, and therefore improve the classification accuracy.

2.2.4 Latent Semantic Indexing

The big and interesting problem with classification of written document data is the amount of data space dimensions. The problem is to generalize the input, to reduce this word-document space. Is there some latent underlying compact information-set in a text document to represent the text with? The Latent Semantic Indexing (LSI) is one approach to discover this idea.

LSI was developed at and patented by Telcordia Technologies, and it was first described in [Dumais et al., 1988][Deerwester et al., 1990]. The interesting with LSI is that it can retrieve relevant documents even when they don't share any words with your query. It decompose the problem using Singular Value Decomposition (SVD), which uncovers the associations among terms in large text collections.

SVD breaks down the original data into linearly independent components. In general many of these components are small and can be ignored, resulting in an approximate model of the data. Each document can then be generally represented with fewer components. Both terms and documents are represented as vectors in a space with reduced dimensions. The dot product between points in the space gives their similarity.

To use LSI we must represent the document as a vector of term frequencies, as the term frequency in section (2.1.1)). Several documents then form a matrix, X_{ij} of word frequencies,

$$X_{ij} = \begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,j} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ f_{i,1} & f_{i,2} & \cdots & f_{i,j} \end{bmatrix}$$

where the index i is the word index and j is the document index. Thus is each word represented of a document vector.

The LSI transforms the matrix to a product of eigenvalues and eigenvectors, a number of linear independent factors. This is called a singular value decomposition of X,

$$X = T_0 S_0 D_0'$$

such that T_0 and D_0 have orthonormal columns and S_0 is diagonal.

If the singular values in S_0 are ordered in size, and if only the k largest ones are kept, we get a approximate of X called $\widehat{X}$,

$$\widehat{X} = TSD'$$

With the reduced diagonal S we are able to do three sorts of comparison:

1. Term-Term: How similar are the terms i and j?
2. Document-Document: How similar are the two documents i and j?
3. Term-Document: How associated are term i and document j?

In [Deerwester et al., 1990] they describe how they have tested the LSI on two text collections, against two straightforward term matching methods. The documents were automatically indexed and terms only occurring in one document were deleted. LSI performed 13% better than the other two systems. One cause to this can be the fact that many test queries are vaguely and poorly stated.

In natural language there are many ways to describe an object, and there are many different meaning of words. This is called *synonymy* and *polysemy*. The LSI can handle problems like synonymy but not polysemy. The problem is that a term cannot have several different positions in the data space.

Dasigi et al. use LSI as a feature extractor and a Back-Prop neural network to integrate the features and classify them [Dasigi and Mann, 1995]. Their goal is to exploit the dimensionality-reduction capability of LSI and the powerful pattern matching and learning capabilities of the neural network. The use of a neural network improves the classifiers accuracy when testing 'new' documents.

The big problem with using LSI is to decide representation dimensionality. In Deerwester et al.'s work they've been guided by "what works best". This is an open issue of research.

2.2.5 Hierarchical indexing

When categorizing it is important to reduce the in-data subset to an optimal subset that gives the best performance. Most researchers do not take into account the hierarchical structure of the vocabulary.

Ruiz and Srinivasan [Ruiz and Srinivasan, 1999] believe that machine learning algorithms could take advantage of these relations and improve performance in text categorization. They have built a system that considers the hierarchical structure of the indexing vocabulary, inspired by a divide-and-conquer model. It divides the problem to smaller problems that are easier to solve, and then combines the solutions to obtain a general solution.

Their system consists of gating networks and expert networks, where the gates are internal nodes and the experts are leaf nodes in a tree structure. The gates decides which nodes to access in the lower levels of the hierarchy, and the experts are specialized in recognizing documents corresponding to specific categories. They use back-propagation neural networks with one hidden layer for both the expert and gating networks, where the hidden layer is twice as big as the input layer.

In the study they make use of a predefined subset of the United Medical Language

System, Medical Subject Headings, to get the relations in the indexing vocabulary.

The system performed slightly, but significantly better than a flat Neural Network Classifier. This is due to that the intermediate layers perform a pre-filtering of "bad candidate texts". Hence the threshold functions in the experts can be set low, without incrementing the number of false positive classifications.

Another research team which have explored the idea of the hierarchical structure of text is Koller and Sahami [Koller and Sahami, 1997]. They point out that the important thing is not the feature selection, but its integration with the hierarchical structure. Then each classifier can use a much small set of 'relevant' features, unlike a flattened system which must consider all features in one step. Also their study shows that a hierarchical classifier performs better than a flat one on this type of data.

Something we should notice with these studies is that they make use of relationship databases for the text. Ruiz and Srinivasan make use of a word relationship database, and Koller and Sahami's text text have been classified with multiple labels. The big disadvantage with hierarchic text classifiers is that one must have access to relationship data for the text and that is not always the case. The problem remains; to find the relationships in text data. I think this is the hard work you want to automate. However, they have taken advantage of the richer model space, something a flat classifier can not do.

2.2.6 Decision trees

When exploring data one may use a decision tree. A decision tree can be used to reduce data volume, into a more compact form, or to discover wheather the data contains well-separated clusters of objects [Murthy, 1997] or not.

The decision tree is constructed as a tree graph with intermediate decision nodes and and leaf nodes. The tree contains zero or more intermediate nodes, and an intermediate node has two or more child nodes. A decision tree decomposes the attribute space into disjoint subsets, using simple rules, which test the data, i.e.:

IF $(a < T)$ **THEN** choose A-child-node **ELSE** choose B-child-node;

The leaf nodes are the classes, the different answers of the decision trees classification of the data.

Constructing a tree from the training is called tree *induction.* There are many ways to do this, and several are *ad hoc* variants of the basic methodology. There are rules derived from distance measures, dependence measures and from the information theory's mutual information and information gain.

ID3 by Quinlan is an algorithm based on entropy measures to find good descriptors for a decision tree. When the data is consistent the resulting decision tree is describing it exactly.

Another algorithm constructed by Quinlan [Quinlan, 1996] is C4.5. C4.5 is an algorithm based on a divide-and-conquer strategy, where the problem gets split up in small pieces using an entropy measure, to select attributes with highest information gain. The attributes (or descriptors) should be representative of the data. Also [Kamber et al., 1997] make use of an entropy measure.

To test an object, you passes it through the tree root node and lets the following intermediate nodes decide which way to follow to a leaf node. When you reach a leaf node the object is classified.

One problem with use of decision trees is the difficulty obtaining a tree with the 'right' size. Some algorithms uses stopping criterions, but the most widely used are *pruning*.

When using pruning you have have constructed a tree where no additional induction improve the accuracy on the training data. Then you remove subtrees which are not contributing significantly to the classification accuracy.

The pruning is considered better than a stopping criterion, because the stopping criterion may stop inducing the tree at a not-so-good node N_1 before reaching a very-good-node N_2. This problem does not arise when using pruning.

Critics point at a weakness of decision trees. The lower levels of the tree we climb the smaller feature sets are used and some of them may not have much probabilistic significance [Murthy, 1997]. Also, several leaf nodes may represent the same class, and resulting in unnecessary large trees. The problems can be solved by fyzzyfication of the data.

2.3 Applications using various techniques for document classification

The number of problems to solve in the field of text classification is large. In this section we will describe a few applications that have been developed to make document classifying decisions. It has been difficult to find such application descriptions, probably because of commercial secrecy aspects. No one gives his good ideas away for nothing.

2.3.1 Letzia

"Letzia is a user interface agent that assists a user browsing the World Wide Web", and is built by Henry Lieberman [Lieberman, 1995]. It operates in tandem with the Web browser and tracks the browsing behavior of the user – follow links, keyword search queries and page idle – and tries to predict what document items may be of interest to the user.

When the user is browsing the Internet, Letzia is browsing too, and explores yet unbrowsed links. At any time, the user can request a set of recommendations from Letzia, based on the current state.

Letzia have no natural language understanding capability, thus browsed pages are only decomposed to lists of keywords. Letzia uses them together with simple heuristics to present the 'best choice'. The goal of Letzia is not preset, it evolvs with the browsing of the user.

When the user follows a link, it indicates that the linked page is interesting in some manner. If the user idles on the page, Letzia believes that the user reads it, and it is added to Letzia's hot-list.

By showing how the user has been browsing Letzia can also explain why it indicates a document as important.

The most common search behavior on the WWW is unfortunately depth-first search. The user misses a lot of information, and finds herself deep in the stack of chosen documents. The use of Letzia compensates this behavior with a breadth-first search, and automatically explores dead ends.

2.3.2 Syskill&Webert

As Letzia, Syskill&Webert, is a software agent that learns to rate web pages, built by Pazzani et al. [Pazzini et al., 1996]. In their work they have tried five different classifiers for the task, including the naive Bayesian classifier, multi-layered Neural Networks and the nearest neighbor algorithm. Their results shows that the naive Bayesian classifier performed the best in most cases.

To be able to classify text the system requires some information from the user. The information is scaled in three points, 'hot', 'lukewarm' and 'cold', and is assigned for each browsed page.

When the user rates a web page, Syskill&Webert saves the document and redoes the document summary of all rated pages. Documents which are used are converted to boolean vectors describing word existence/nonexistence in the text.

The agent is also able to form a LYCOS query, to provide the user with interesting links. It does this using the 'hot'-document words. Syskill&Webert filters out ordinary English words, using mutual information. Since LYCOS can not accept long queries the agent uses the seven most discriminating words.

[Pazzini et al., 1996] found out that users of their agent did not read the entire pages before rating them. This results in errors when Syskill&Webert analyses too much of the document. In a patched version they considered this. The new system classifies only the beginning of the pages, and outperforms its precursor.

In [Billbus and Pazzini, 1996b] they extend the Syskill&Webert agent. The user is able to feed the system with interesting words. This extended system outperformed the original one. They think that the extra information cannot be extracted automatically from the training set by statistical methods alone.

2.3.3 MailCat

When receiving mail many users sort it into folders. Typical for mail-reading applications is to provide a long list of existing mail-folders. Sorting work is tedious and is in many cases undiscouraging users from filing their mail in a manageable way. MailCat described in [Segal and Kephart, 1999b] offers aid to this task. MailCat predicts the most suitable mail-folders for the incoming new mail, and makes a smaller set of choices avaliable, three folders which the user can choose among. In 80 to 90% of the cases MailCat provides the right folder.

MailCat offers this without demanding anything in return. When MailCat is installed it analyses the existing folders and construct a classifier for each one of them.

The used classifier represent each text as a word-frequency vector, and each folder as a weighted word-frequency vector. The similarity between the test text and a folder is computed as a distance between text and folder vectors. Unfortunately this task is time consuming due to the vector size, which can grow to ten megabytes or more. Because of this [Segal and Kephart, 1999b] make use of a cosine distance, proposed by Gerald Salton and Michael J. McGill, called *SIM4*, involving only words in the test text, not the whole word space.

In [Segal and Kephart, 1999a] they make more detailed tests. They examine how the system reacts to new users, new information, new folders and how important incremental learning is. One interesting behavior was the inverted learning curve when introducing new folders. MailCat makes a bad choice and drives the learning curve down. The system is not to blame when a new message is classified incorrectly because of the absence of an appropriate folder. In the beginning this behavior is frequent, but the more information it gets and the more folders created, the behavior get less frequent. The learning curve takes on the common characteristic shape.

2.3.4 NewsDude

Most IR systems assume that the user has a specific, well-defined information need. But that is not always the case. Instead, the user query could be phrased as: "What is new in the world that I do not yet know about, but should know?".

Billbus and Pazzini [Billbus and Pazzini, 1996a] describe a system called NewsDude which takes care of the users long-term interests and short-term interests. This is handled by a hybrid-model, where the short-term memory is based on a k-nearest-neighbor algorithm and the long-term memory is based on a naive Bayesian classifier.

The classifier tries to classify the text it with the short-term memory, and if that fails it tries to classify it with the long-term memory.

This system can handle three characteristics of a user a 'non-hybrid' system can not:

1. Multiple interests of the user.
2. Quickly adapt to a user's changing interests.
3. Change of the user interests as a direct result of interaction with information.

The third characteristic have not received any attention in the IR community.

2.3.5 The Fast Search Engine

The enormous size of the Internet demands search services of different types. Described in [Fast, 1998a][Fast, 1998b] is a Norwegian web search engine called The Fast Search Engine. They claim that it is one of the best search engines on the net, indexing 300 Million non duplicated documents (January 2000). The Basis of the system is the FAST Pattern Matching Chip (PMC), which is combined with FAST's state of the art search algorithm, FAST SW Search [Fast, 1998b]. This system handles linguistic problems like stemming and approximating words, and boolean operators.

Chapter 3

The Bayesian Artificial Neural Network

In this section we describe the theoretics of the Bayesian Artificial Neural Network. We begin with the Artificial Neural Network topology. Then we describe the naive Bayesian classifier, and use these two to build a one-layer Bayesian Neural Network. At the end we describe how to extend the idea with hidden columns to construct a multi-layered Bayesian Network.

3.1 The signal flow of an Artificial Neural Network

The Artificial Neural network is generally described in section (2.2.1). The signal forwarding properties of the neural network is modeled as a weighted sum of the input signals,

$$s_j = b_j + \sum_i w_{ij} o_i \tag{3.1}$$

where b_j is the bias and w_{ij} is weights between input signal i and neuron j. The signal is passed through an activation-function ψ,

$$o_j = \psi(s_j)$$

The activation function is often non-linear and anti-symmetric. See figure (3.1) for the artificial neuron topology. The neuron splits the input space into two subsets with a hyper-plane, where input from one subset activates the neuron and input from the other subset inactivates it.

In an Artificial Neural Network the neurons are connected to each other. There are several suitable topological solutions/architectures for solving different problems. A commonly used architecture is a layer architecture, where the neurons are placed in layers. The neurons in one layer feed the neurons in the next layer. An ANN with this architecture is called a *feed-forward network*, see figure (3.1). The learning capabilities of the artificial neuron is handled with in the learning-

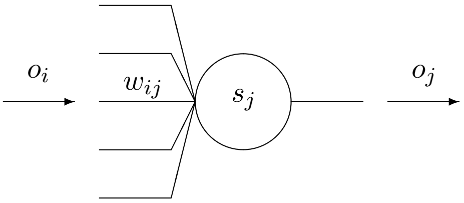

Figure 3.1. The Neuron

algorithm. There are several learning-algorithms, but the most commonly used is the *back-propagation rule*. In the Back-Prop rule the output signal of the network, o_j, and an expected output signal, d_j, is compared and results in an error signal of the network, $e_j = o_j - d_j$. Naturally, the error is dependent of the weights in the network, and the learning-rule lets us change them in direction to a smaller error.

One disadvantage with the ANN is that it requires lots of training data. The more data you have the better generalization possibilities you get. Reinforcement learning is one way to get around this problem. But in the other hand you have to construct a representative environment for the network to 'live' in. Also it has to seek through the whole state-space to get the ANN fully trained, which can be very time consuming.

3.2 The naive Bayesian classifier

The task of a classifiers is to classify a set of inputs into a class in a set of classes. For every input we want to output the most probable class. Thus, we have to calculate the probability for every possible output. If we output the class with highest

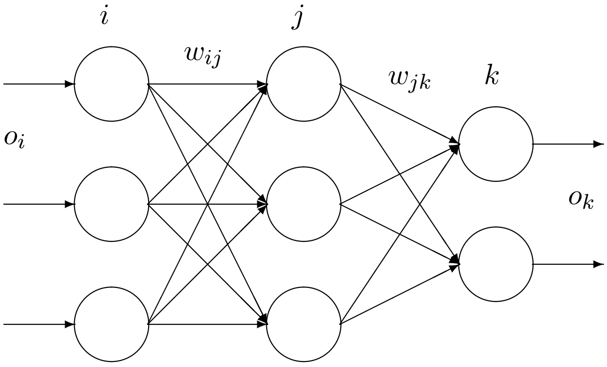

Figure 3.2. A feed-forward multi-layered Neural Network with one hidden layer.

probability we will minimize the number of errors.

Thus, given one input x we want to calculate the probability for y, $P(y|x)$. This can be done by the rule of Bayes Conditional probability:

$$P(y|x) = P(y)\frac{P(x|y)}{P(y)} \tag{3.2}$$

It is often easier to express the looks of a class y, in the meaning of the attribute x, instead of expressing the looks of attribute x in the meaning of class y.

3.2.1 The independence assumption

If we have N independent input attributes $\overline{x} = \{x_1, x_2, \ldots, x_N\}$ we can calculate the joint probability $P(\overline{x})$ as:

$$P(\overline{x}) = P(x_1)P(x_2)\ldots P(x_N) \tag{3.3}$$

The conditional probability of $\overline{x}$ given class y becomes:

$$P(\overline{x}|y) = P(x_1|y)P(x_2|y)\ldots P(x_N|y) \tag{3.4}$$

With (3.3) and (3.4) inserted in (3.2) we get $P(y|\overline{x})$:

$$P(y|\overline{x}) = P(y)\frac{P(\overline{x}|y)}{P(\overline{x})} = P(y)\prod_{i=1}^{N}\left(\frac{P(x_i|y)}{P(x_i)}\right) \tag{3.5}$$

This is the basis of the naive Bayesian classifier.

3.3 The one-layer Bayesian Neural Network

Equation (3.5) underlies the concept of the Bayesian Neural Networks (BANN) [Lansner and Ekeberg, 1989]. Yet, one can not see the similarity with the signal summation (3.1) of the ANN, but if we take the logarithm of (3.5) it can be written as a sum:

$$\log P(y|x_i) = \log P(y) + \sum_{i=1}^{N}\log\left(\frac{P(x_i|y)}{P(x_i)}\right) = \log P(y) + \sum_{i=1}^{N}\log\left(\frac{P(y,x_i)}{P(y)P(x_i)}\right) \tag{3.6}$$

If we, given a set of observed inputs $A = \{x_i, x_j, x_k, \ldots\}$, want to calculate the probability of a specific outcome y we write equation (3.6) as:

$$\log P(y|A) = \log P(y) + \sum_{x_i \in A} \log \left(\frac{P(y, x_i)}{P(y)P(x_i)} \right) = \log P(y) + \sum_i \log \left(\frac{P(y, x_i)}{P(y)P(x_i)} \right) o_j \tag{3.7}$$

When comparing (3.7) with (3.1) we see that,

$$b_j = \log P(y) \tag{3.8}$$

$$w_{ij} = \log \frac{P(y|x_i)}{P(y)} \tag{3.9}$$

If we have observed M classes $\overline{y} = \{y_1, y_2, \ldots, y_M\}$ we have a class index too:

$$b_j = \log P(y_j) \tag{3.10}$$

$$w_{ij} = \log \frac{P(y_j, x_i)}{P(y_j)P(x_i)} \tag{3.11}$$

$$s_j = b_j + \sum_i w_{ij} x_i \tag{3.12}$$

To prevent probablitities greater than 1 we normalize the output

$$x_j = \frac{exp(s_j)}{\sum_j exp(s_j)} \tag{3.13}$$

3.3.1 Training the one-layer Bayesian Neural Network

To train the one-layered BANN we will not use any algorithm alike the feed-forward algorithm. The feed-forward algorithm uses the input data several times during the training, but in the Bayesian learning rule only once.

The training of the BANN includes estimating the probabilities for attribute occurences, class occurences and the joint occurences of attribute and class. To estimate the probabilities we need to count the occurences in the training sets. The occurrence counters are C - the total number of training patterns,

$$C = \sum_p \kappa_p \tag{3.14}$$

c_i - the total occurrences of unit i,

$$c_i = \sum_p \kappa_p \xi_{i,p} \tag{3.15}$$

c_{ij} - the total simultaneous occurrences of unit i and unit j,

$$c_{ij} = \sum_p \kappa_p \xi_{i,p} \xi_{j,p} \tag{3.16}$$

where the κ_p is the strength of pattern number p, and $\xi_{i,p}$ indicates the presence of attribute i in pattern p.

Now we can calculate the probability for all the interesting occurrences P_i, P_j and P_{ij}.

$$P_i = \frac{c_i}{C} \tag{3.17}$$

$$P_{ij} = \frac{c_{ij}}{C} \tag{3.18}$$

These are the classical probability estimations. If we use these definitions we get problems with logarithm of zero in equations (3.10) and (3.11) as some combinations never occur in the training set. Even the ability to generalize gets weak. Lets look at an example!

> **Example:** Think of a 6-eyed dice. If we throw the dice six times, it is very unlikely to get one six, one five, one four and so on. It is very unpredictable to use this small test to estimate the classical probability. But if we throw it a hundred times we will be more able to trust the statistical outcome. The bigger training set the better statistics.

3.3.2 The Bayesian factor

To solve the logarithm-of-zero problem we introduce the Bayesian approach:

$$P_i = \frac{c_i + \frac{\alpha}{n_i}}{C + \alpha} \tag{3.19}$$

$$P_{ij} = \frac{c_{ij} + \frac{\alpha}{n_i m_j}}{C + \alpha} \tag{3.20}$$

where α is the Bayesian factor, and can be thought of how much importance we lay on c_i, the n_i factor is the number of possible outcomes of X_i and m_j is the number of possible outcomes of Y_j [Holst, 1997].

The value of α spans between 0 and 1, and is normally set low; $\alpha = 1/C$ is considered to be good [Holst, 1997]. These probablities work better in small test sets than their classic alternatives, as they take in acount the number of possible outcomes of each attribute.

There is also a second approach, where the probablitity $P(y|x_i)$ used in (3.9) is:

$$P(y|x_i) = \frac{c_{ij} + \alpha P_j}{c_i + \alpha} \tag{3.21}$$

3.4 The multi-layer Bayesian Neural Network

In the above one-layered BANN we assumed all input attributes are independent. But when they are not the above naive approach is not appropriate. To improve performance we can introduce a hidden middle layer of units. Those are are grouped in so called *Complex columns.* The columns can be constructed in several ways. The original construction is the Partitioning Complex column, and two other are the Overlapping Complex column, Fragmented Complex column.

All the Complex columns express the activation of different combinations of units in the input layer.

3.4.1 Partitioning complex columns

If the two inputs X_i and X_j are dependent we can not use equation (3.3). We will end up with a unresolvable expression like this:

$$P(\overline{x}) = P(x_1)P(x_2)\dots P(x_i, x_j)\dots P(x_N) \tag{3.22}$$

For those inputs which are dependent, we would like to consider the joint probability $P(X_i, X_j)$ rather than the product of the marginal probabilities, $P(X_i)P(X_j)$. In the partitioning complex columns (Fig 3.3) the input attributes are grouped (partitioned) in M independent groups, read *complex columns.* We introduce a 'joint' variable U_k for each of these groups. Now we can express the conditional joint

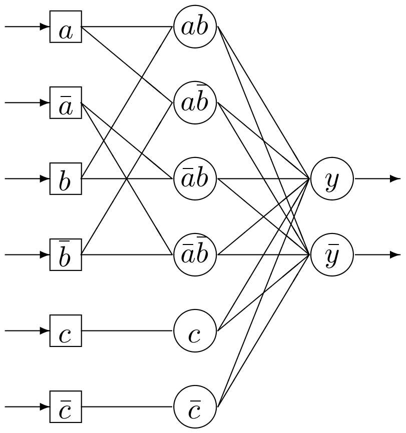

Figure 3.3. Partitioning complex column network. Here are the attributes a and b dependent and grouped in a complex column.

probability equation 3.5 as:

$$P(y|\overline{x}) = P(y)\frac{P(\overline{x}|y)}{P(\overline{x})} = P(y)\prod_{i=1}^{M}\left(\frac{P(u_i|y)}{P(u_i)}\right) \tag{3.23}$$

Training the partitioning complex column

The partitioning complex column network is trained as the single layered network, considering the complex columns rather then the original input nodes (3.11).

$$w_{ij} = \log\frac{P(u_i|y)}{P(u_i)} \tag{3.24}$$

The problem with this network is that its size increases exponentially. One more severe problem is that the ability to generalize decreases. The more unique/specialized features you consider as input the less ability to generalize.

3.4.2 Overlapping complex columns

When using partitioning complex columns you are forced to construct combinations of attributes that are not correlated.

Suppose that attribute X_1 and X_2 are correlated, and attribute X_2 and X_3 are correlated, but attribute X_1 and X_3 are not correlated. In the partitioning complex column solution we have to build all combinatorial combinations of the three attributes. This should not be necessary; we have no X_1X_3 combination to compensate for. It would be nice to only construct complex columns for the combinations X_1X_2 and X_2X_3, but those two columns would be dependent through X_2. According to A. Holst there is a way to handle this problem [Holst, 1997]. All joint probabilities can be written as a product through the chain-rule:

$$P(\overline{x}) = P(x_1)P(x_2|x_1)P(x_3|x_2,x_1)\ldots P(x_N|x_{N-1},\ldots,x_2,x_1) \tag{3.25}$$

But if we have no cyclic dependencies we can write a variable to depend only on the variables it is conditioned on.

Example: If A, B and C are stochastic variables and C depends on B, and is independent of A, we can make the following rewriting:

$$P(C|B,A) = P(C|B)$$

Thus we can write the variable to only be conditioned on the variables it depends on.

We use this on equation (3.25) and get:

$$P(\overline{x}) = \prod_{i=1}^{N} P(x_i|S_i) = \prod_{i=1}^{N} \frac{P(x_i,S_i)}{P(S_i)} \tag{3.26}$$

where S_i is the set of variables x_i directly depends on, and

$$P(S_i) = \prod_{j:X_j\in S_i} P(x_j). \tag{3.27}$$

Inhibition

When using overlapping complex columns we have to compensate for the activation of both the combination and the original input nodes. This is done by inhibiting the input nodes. This have to be done not only by inactivating the nodes but by subtracting one unit of activation from them. So, if a node helps to activate several combinations it will be heavily inhibited and eventually to getting a negative activity.

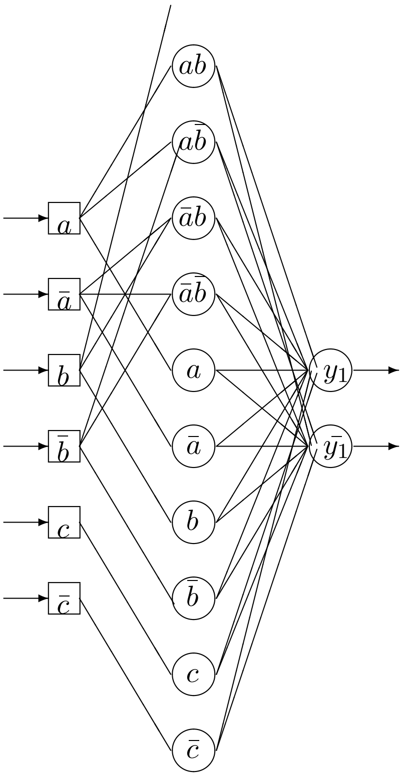

Figure 3.4. Overlapping complex column network.

3.4.3 Fragmented complex columns

Using partitioning complex columns or overlapping complex columns (described in the sections (3.4.1) and (3.4.2)) with large input attributes, results in a huge amount of new combination units. Many of the combined inputs might be uncorrelated and it would be unnecessary to build those combinations. They may even worsen the capability of the network. It would be interesting to create only combinations which improved the results of the network.

Suppose that we have two practically uncorrelated attributes a and b, with two units each, **yes** and **no**. Then we do not have to merge them. But if the **yes-yes**-combination of a and b is correlated, we would have to build the that combination to get better classification performance for this case. A network without this node would perform in all cases but the **yes-yes** case.

In figure (3.5) is we see an example of a fragmented column network, build to solve the described scenario.

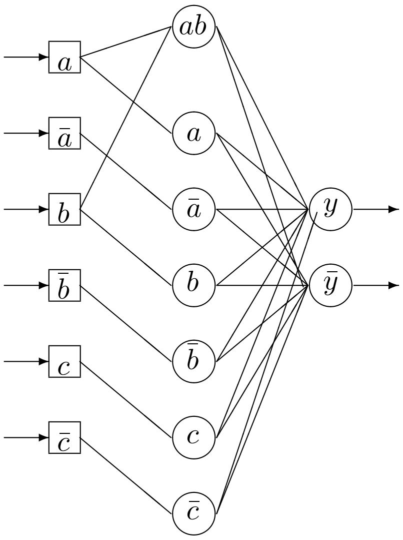

Figure 3.5. Fragmented complex column network

Inhibition

The inhibition used in fragmented complex columns are the same as for overlapped complex columns.

3.4.4 Choosing columns

When constructing complex columns in the multi-layer Bayesian Network we might not want to expand all node dependencies. We would like to construct only the most 'effective' columns, described in section (**??**). One idea is to merge those attributes that correlates and construct a joint column for them.

There are several different measures to use when dealing with correlations. In this kind of problem, the most commonly used is Mutual Information (2.7).

We are interested in creating complex columns for the attribute combinations with the highest mutual information. They are the ones with the highest dependencies, and combining those should improve the classifier.

Chapter 4

Document classification with a Bayesian Artificial Neural Network

This chapter will discuss automatic text classification with a system based on the naive Bayesian classifier (section 3.2). The problem is defined in section (4.1). The tests are done on a text set containing user rated conference calls from the work of Waern and Rudström [Waern and Rudström, 2000]. Their test setup is briefly described, along with the structure of the document data. The method is described in section (4.2).

The tests are made in different steps. In section (4.3) one-layer Bayesian network from section (3.3) is used. Section (4.4) introduces the use of multi-layer Bayesian neural network from section (3.4).

Different ways of reducing the size of the input data is tested and if it is interesting to cluster words in combinations.

4.1 Definition of the problem

A person who has research related work often gets information about forthcoming conferences in different topics. However, it is unusual that she is interested in all these conference calls.

The problem is to help this person to filter out calls that she is not interested in, to prevent that she gets showered by all these calls. This task can be generalized to filtering of information.

With the information of which documents the specific person likes and dislikes the expectation is to find document features which indicates that the document is interesting or uninteresting, and thus be able to build an interest profile for the specific person.

In the extent several user-profiles can be seen as one users several different topics of interest.

4.1.1 Waern and Rudström

The database used in this survey origin from the work of Waern and Rudström [Waern and Rudström, 2000]. The reason why choosing this data was just to have some data and results from SICS to compare to.

Waern and Ruström made three different test setups:

1 . In the 'original profiles' setup the reader has created her profile on the basis of annotated documents. The setup simulates a new system where almost no training had been done.

2 . In the second setup a non standard algorithm was used to move the user initial profile 'closer to' or 'farer from' the document profile. When the user and document profiles are moved, they 'inherit' terms from each other. Eventually the documents have values associated to most of the terms used in reader profiles or other document profiles. This is the 'modified profiles' setup which is a simulation of a 'stable system' situation.

3 . The 'generated profiles' results was achieved in the same matter as the 'modified profiles' but where the user profiles where generated from scratch.

The results from the three setups can be found in table (4.1).

	Average precision	Average recall
Original profiles	21%	70%
Modified profiles	35%	77%
Generated profiles	31%	62%

Table 4.1. Precision and recall for three system setups.

4.1.2 Document representation

The database contains 84 documents in form of conference calls. The documents have been parsed through a program that adds linguistic properties to the text (as stems and information about inflection), and outputed in a SGML format, where the document is broken down into tokens. The parser is written by Jussi Karlgren at SICS.

Example SGML-token:

```
<TOKEN SPAN="871-882" NBR="140" SF="researchers"
       LEMMA="researcher" POS="N" NUM="PL" CASE="NOM"
       SYN="@<P" AF="<DER:er>">researchers
</TOKEN>
```

Here the **SPAN** is the character position span in the text, **NBR** the word number in the text, **SF** the full form, **LEMMA** the common case, **POS** the part of speech, **NUM** the numerus (plural or singular), **CASE** the case, **SYN** the syntactic function and **AF** features for the specific part of speech.

A word token might have several analysises. This is due to the fact that a word without context is not descriptive enough. In this thesis work I do not make use of any fancy process to choose which analysis to allot the specific token. I just pick the first one and this may be a source of error.

The test system used during the test utilizes only the **LEMMA** part of the analysis. Due to the amount of lemmas in a text is less or equal the amount of unique words this reduces the dimension of the in-data, and hopefully the in-data gets more generalizable.

Each unique stem in a document forms a boolean input attribute to the classifier.

1 - the occurrence of the stem (yes-node)

0 - non-occurrence of the stem (no-node)

The term frequency tf is not considered. One document forms a boolean vector of unique stem occurrences and non-occurrences. This vector is feed as input to the classifier.

4.1.3 User information

The described documents have been ranked by 25 test users as a part of the work from [Waern and Rudström, 2000]. This ranking have two values, **save** or **throw**, meaning that the user likes or dislike one specific document. At the training and testing stages, the user ranking is used as the expected value for the classifier.

4.2 Method

Several tests will be performed. Is there possible to delete attribute 'noise'? How should the Bayesian factor α be set? Is there any need for representing each input attribute with two nodes? Which approach is suitable when defining the probability estimate of P_{ij}? Do use of complex columns improve the prediction?

I am using MATLAB for all calculations, and plots.

4.2.1 Testing method

One document vector and the user ranking for that document is considered as one training or testing example. All tests are made by a *leave-one-out*-test, even called *cross validation* test, where all example-data except one item is used for training, and the test is performed on the left out item.

It is very important not to test on the training examples. The feature of the system one would like to investigate is the systems ability of making generalizations, not its memory capacity.

4.2.2 Result explanations

When analyzing the test results, it is important to use a good measure which can indicate how effective the tested system is. At first one might count the number of correct predictions (save and throw) and divide it by the total number of predictions like this:

$$correct\ fraction = \frac{correct\ number\ of\ predicitions}{total\ number\ of\ predicitions} \tag{4.1}$$

That is not a fair measure, though. If the user wants few documents the system may predict that the user does not want any documents at all. This situation will result in a high correct-fraction-value, because of the many correct **throw**-predictions. But this does not mean that the system is behaving properly; the user does not receive anything.

The result plots and tables in this thesis are showing mean-values of **mutual information**, **precision**, **recall** and **correct fraction** calculated over all classes/users and all test documents. We could also look at the median and variance of the given measures, but I think the mean-value is enough to get an understanding of how the system behaves.

The different measures are always using the same type of line in the plots: Mutual information - Solid (-), Precision - Dotted (:), Recall - Dashdot (:-), Correct fraction - Dashed (–).

In some tests the mean-values are **NaN** (not a number). This occur when the measure contain a division by zero. The meassure gets undefined, NaN.

4.3 Using a one-layer Bayesian Artificial Neural Network

This section describes the tests performed with a one-layer BANN. We will test if reducing the attribute vector size improves the classifier, how the Bayesian factor should be choosen, if there is a need of two units per attribute and which approach of the Bayesian probability P_{ij} to use.

We will also inspect the attribute weights and train on the training set to minimize the risk of human errors.

4.3.1 Reference classifier

Before screwing and fixing any parameters a reference system will be set up which we may compare the different modified systems to. The reference system consists of a one-layer BANN for each user, where the input attributes are represented with two units. The Bayesian parameter α is set as:

$$\alpha = \frac{1}{C}$$

where:

$$C = \{number\ of\ training\ documents\}$$

Results

The mutual information of the classifier tends to get higher the more documents the users wants (see figure 4.1). The test data is quite small and if the wanted documents are few their attribute contribution to the classifier will be drowned by the not wanted documents.

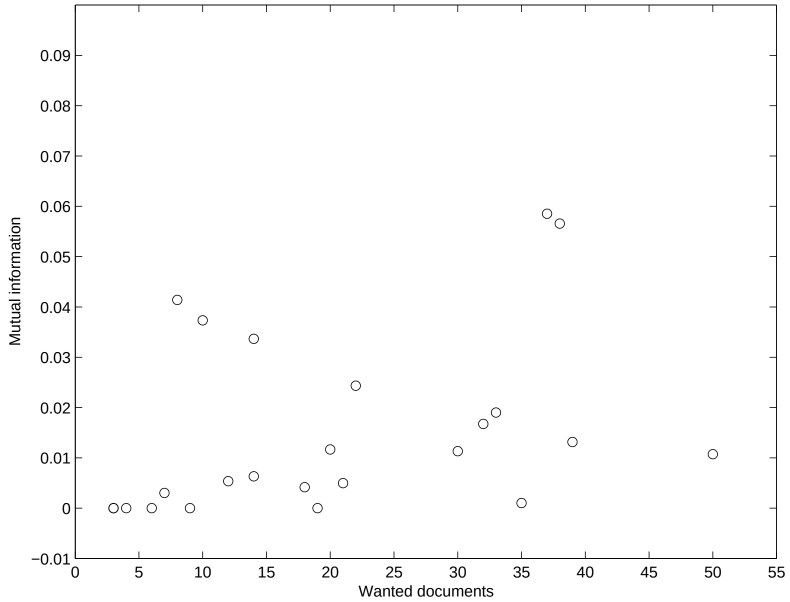

Figure 4.1. Mutual information between the classifier prediction and the expected prediction as a function of wanted documents. Each point represent one user.

System	Correct fraction	Mutual information	Precision	Recall
One layer BANN	0.9881	0.0178	0.4408	0.0490
Waern Original profiles			0.21	0.70
Waern Modified profiles			0.35	0.77
Waern Generated profiles			0.31	0.62

Table 4.2. Reference results of a Bayesian Artificial Neural Network with $alpha = 1/C$ and two units per attribute, and the classifers from Waern and Rudström [Waern and Rudström, 2000].

The reference system behavior is shown in table (4.2). This is a bad result compared to the results from [Waern and Rudström, 2000] where the recall is much higher, 72% or 62%.

4.3.2 Noise reduction

A document usually contains many words which are irrelevant to the reader. When it comes to classifying documents these words can probably be considered as noise, and would not provide any useful information or patterns for the classifier. Is it possible to get good results when pre-filtering out these words from the document representation?

High frequent words as 'and' and 'or' are probably not as important as the moderate frequent words 'artificial' and 'intelligence'. The classifier would give unimportant words would low weights, and we should be able to erase them, to reduce the input dimensionality, without decreasing the performance.

Low frequent words may not be good features to a text either. Depending on the Bayesian constant α, these words would get a weight near zero. In the extent, a word which occurs only once in the data set does not help the classifier at all, because the word occurs either only in the training-set **or** only in the test-set. To be a support for the classifier the word has to exist in both the training set and the test set.

Before we pre-filter the text we have to investigate how frequent words are in the texts. Figure (4.2) shows that most of the words occur in less than 10 documents.

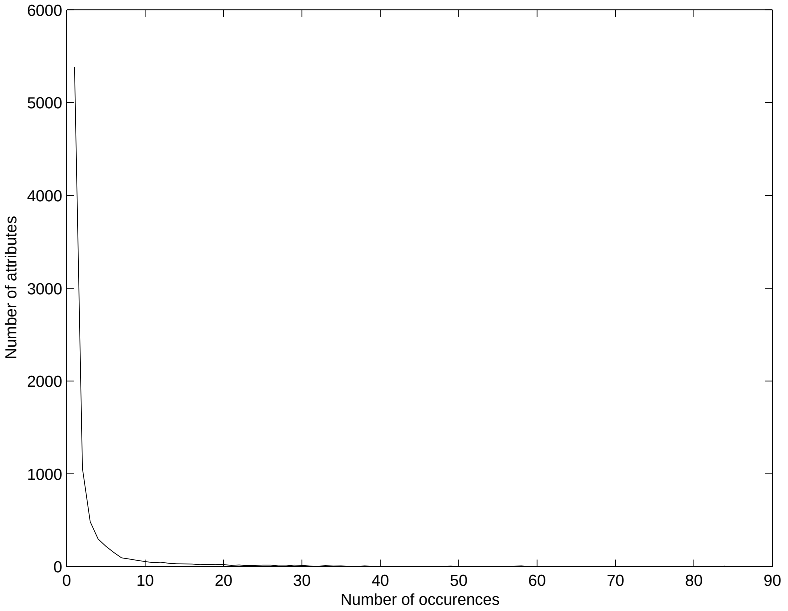

Figure 4.2. Attribute occurrences.

In this section, the supposed noisy text set will be pre-filtered. The low frequent and the high frequent words are erased from the input vector.

Results

The results is shown in figure (4.3).

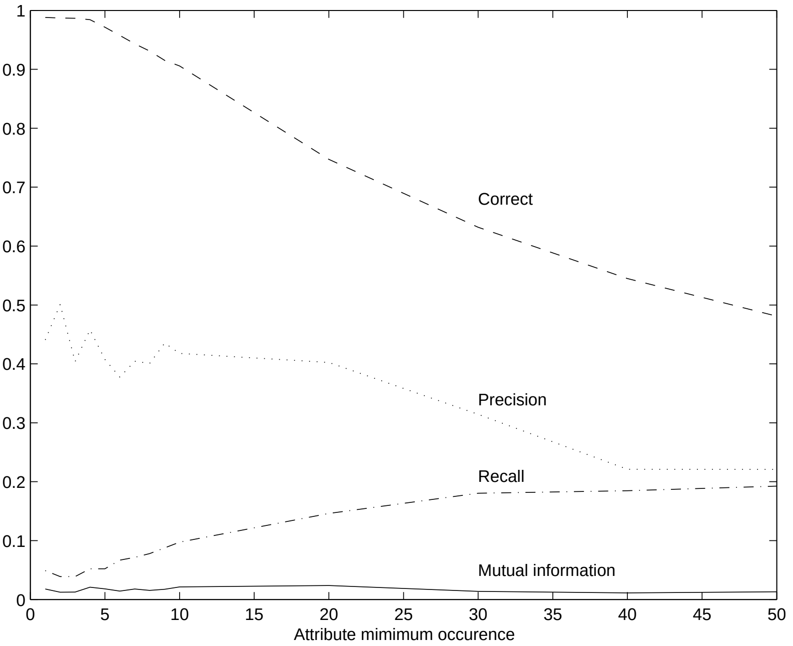

Figure 4.3. Minimum attribute occurrences. Values for this plot can be found in Table (4.3).

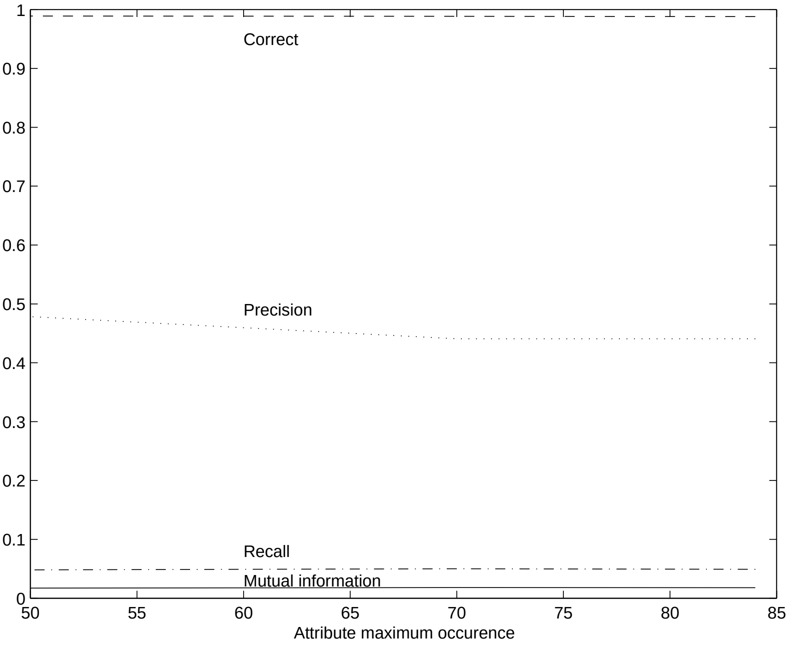

Figure 4.4. Maximum attribute occurrences. Values for this plot can be found in Table (4.4).

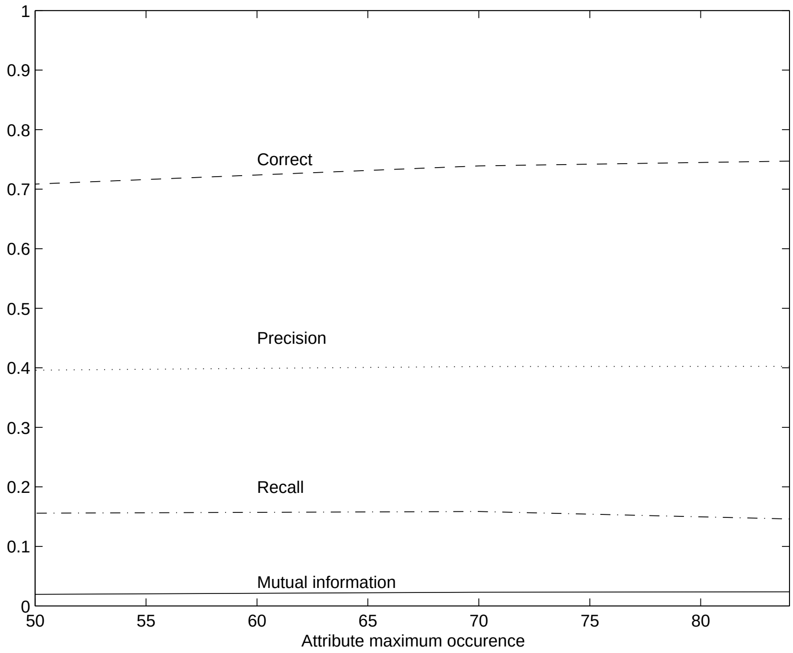

Figure 4.5. Maximum attribute occurrences when minimum occurence is 20. Values for this plot can be found in Table (4.4).

Attribute min occurence	Correct fraction	Mutual information	Precision	Recall
1	0.9881	0.0178	0.4408	0.0490
2	0.9871	0.0125	0.5008	0.0389
3	0.9867	0.0126	0.4039	0.0388
4	0.9843	0.0209	0.4577	0.0522
5	0.9714	0.0182	0.4076	0.0521
6	0.9576	0.0143	0.3776	0.0669
7	0.9433	0.0178	0.4045	0.0717
8	0.9310	0.0155	0.4007	0.0780
9	0.9152	0.0173	0.4354	0.0877
10	0.9057	0.0215	0.4178	0.0973
20	0.7471	**0.0238**	0.4025	0.1462
30	0.6319	0.0138	0.3144	0.1804
40	0.5448	0.0111	0.2212	0.1847
50	0.4810	0.0132	0.2208	0.1926

Table 4.3. Results from deleting lowfrequent-noise attributes.

Conclusions

If we concentrate on the mutual information results, it seems like filtering out attributes that occur in less than ten document is a good idea. It is not a big result

Attr min occurence	Attr max occurence	Correct fraction	Mutual information	Precision	Recall
1	84	0.9881	0.0178	0.4408	0.0490
1	70	0.9886	0.0181	0.4408	0.0500
1	50	0.9890	0.0172	0.4784	0.0482
20	84	0.7471	0.0238	0.4025	0.1462
20	70	0.7390	0.0232	0.4020	0.1586
20	50	0.7086	0.0194	0.3960	0.1559

Table 4.4. Results from deleting highfrequent-noise attributes.

increase, but as we are using both **occur**- and **not-occur**-nodes the abcense of many low frequent attributes will not contribute to the classifier.

Why is it, that deleting words that only occur once, in the training set or in the test set, gets us worse results than not deleting them? At the first glance it should not make any difference. Actually it does! Suppose the user wants less than 50% of the documents. If one attribute only occur in the test set the probability of belonging to the class is higher not belonging to it.

4.3.3 Bayesian factor selection

The classification is dependent on the Bayesian parameter α. The Bayesian factor value is normally set low, as described in section (3.3.2); the value $1/C$ is considered good. I think that a good alpha is dependent of the characteristics of the system environment. It could also be depentant of the different user behaviors, but without any rigid method to set the factor on the basis of user information, we can not do anything else than picking the overall best value. In this section we test the dependency of α. We will test different alphas with and without deleting low frequent attributes.

Results

For your curiosity I also present some mutual information plots for the different user classifiers separated, see plots (4.8) and (4.9).

Conclusions

First, just by comparing the two plots we see that deleting low frequent attributes was a bad idea. Probably we delete to many low frequent attributes used by the classifiers. Hence, we should not delete any attributes.

Figure (4.7) together with table (4.6) shows that $0.22 \leq \alpha \leq 0.50$ is the best we get. Here the mutual information is as high as possible, the recall flattens at

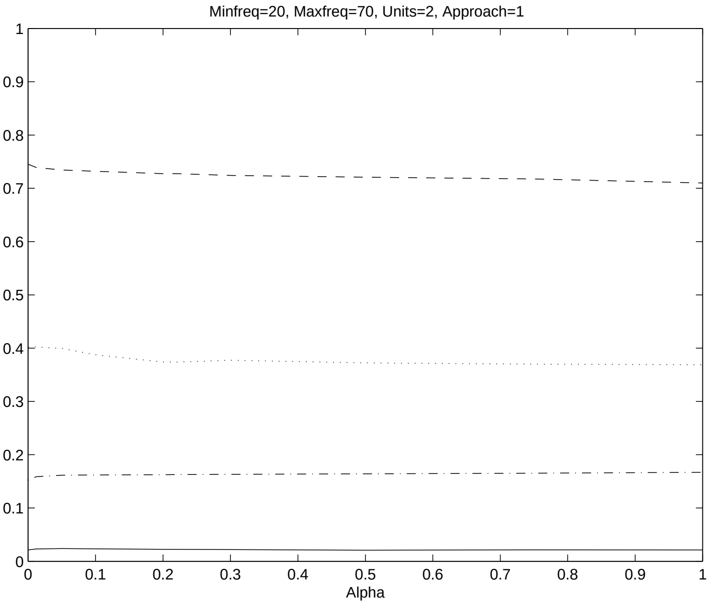

Figure 4.6. Different alphas when minimum term occurence is 20 and maximum occurence is 70.

Alpha	Correct fraction	Mutual information	Precision	Recall
0.0001	0.7452	0.0211	0.4021	0.1520
1/C=0.012	0.7390	0.0232	0.4020	0.1586
0.05	0.7343	0.0239	0.3997	0.1616
0.10	0.7319	0.0232	0.3874	0.1619
0.20	0.7276	0.0224	0.3741	0.1625
0.22	0.7276	0.0224	0.3739	0.1626
0.30	0.7243	0.0219	0.3773	0.1631
0.50	0.7210	0.0208	0.3723	0.1640
0.75	0.7176	0.0213	0.3699	0.1654
1.00	0.7100	0.0211	0.3687	0.1669

Table 4.5. Results from using different alphas when minimum term occurence is 20 and maximum occurence is 70

$\alpha = 0.3$, and the precision is falling. At the refered interval we got the best combination. Compared to Waern and Rudströms work (table 4.1) it is a higher precision but a lower recall. High recall is a must, but if the precision does not get higher

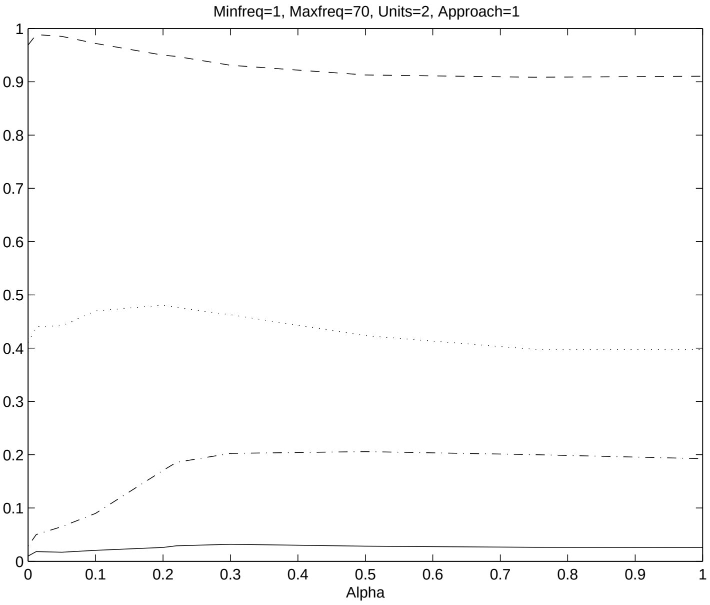

Figure 4.7. Different alphas, but without erasing low frequent terms.

Alpha	Correct fraction	Mutual information	Precision	Recall
0.0001	0.9690	0.0098	0.4101	0.0285
1/C=0.012	0.9886	0.0181	0.4408	0.0500
0.05	0.9852	0.0170	0.4418	0.0650
0.10	0.9719	0.0206	0.4698	0.0899
0.20	0.9500	0.0259	0.4806	0.1706
0.22	0.9476	0.0291	0.4762	0.1855
0.30	0.9310	**0.0319**	0.4627	0.2025
0.50	0.9129	0.0283	0.4234	0.2058
0.75	0.9086	0.0262	0.3979	0.2001
1.00	0.9105	0.0258	0.3973	0.1924

Table 4.6. Results from using different alphas when minimum term occurence is 1 and maximum occurence is 70

than $precision_{min}$ given by:

$$precision_{min} = \frac{relret}{ret_{max}} = \frac{number\ of\ wanted\ documents}{total\ number\ of\ documents} \tag{4.2}$$

$$= \frac{490}{2100} = 0.2333 \tag{4.3}$$

the behavior of the is equal to not using any filter at all! Of course we would like to use a filter system which performs better than that.

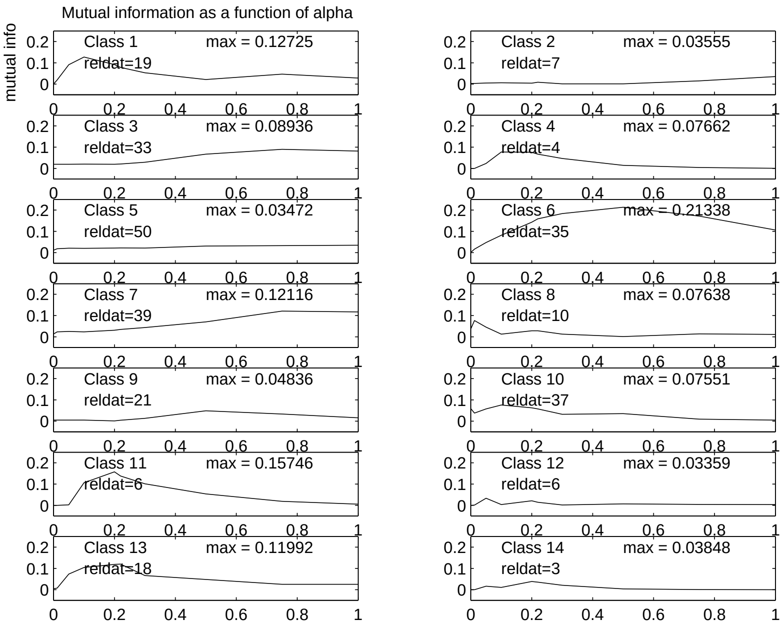

Figure 4.8. Mutual information depending on the Bayesian factor. Classes 1 to 14.

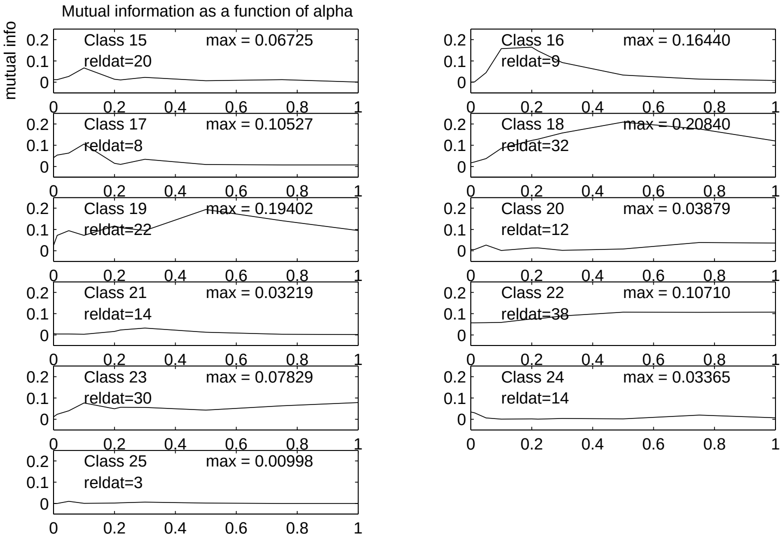

Figure 4.9. Mutual information depending on the Bayesian factor. Classes 15 to 25.

If we look at the plots (4.8) and (4.9) we can not find any pattern in the diffferent classes dependence of alpha.

4.3.4 One or two nodes per attribute

Do we need two nodes per attribute, (**occur** and **not-occur**), or is it enough with only an **occur**-node? I think that the occurrence of a word is more interesting than the non-occurrence of a word.

If that assumption is correct, it would be unnecessary to represent the non-occurrence of words in the document vector and one could decrease the memory consumption of the classification system. In this section a one-unit-per-attribute setup will be tested in the same manner as the two-unit-per-attribute setup in section (4.3.3).

Results

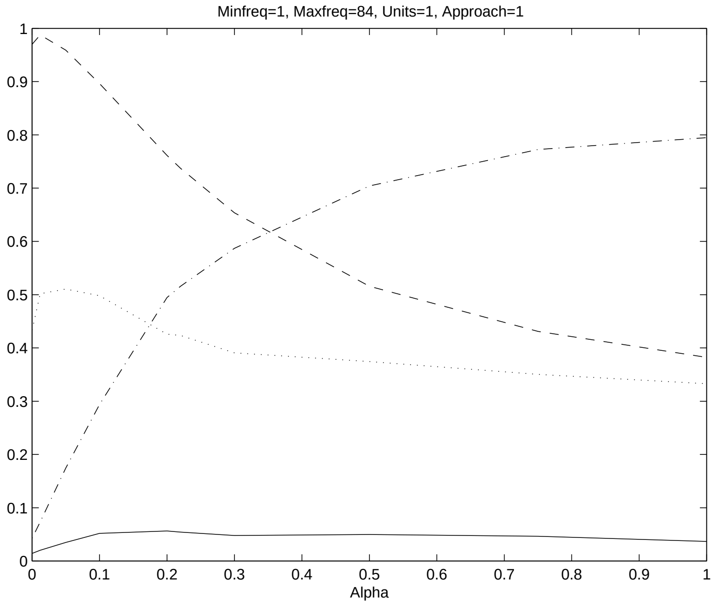

Figure 4.10. One unit per attribute, and maximum occurence 84. Measures for the plot can be found in table (4.7).

Conclusions

When comparing the two plots (4.10) and (4.11), where the maximum occurence is 84 and 70, we see that deleting high frequent attributes does not improve the classifier.

Comparing these results to the result from section (4.3.3) we see that one unit per word is to prefer. The precision is slightly worse, 42% compared to 46%, but

Alpha	Correct fraction	Mutual information	Precision	Recall
0.0001	0.9700	0.0144	0.4334	0.0430
1/C=0.012	0.9876	0.0202	0.5017	0.0734
0.05	0.9590	0.0350	0.5104	0.1744
0.10	0.8962	0.0520	0.4979	0.2938
0.20	0.7614	**0.0564**	0.4258	0.4948
0.22	0.7371	0.0544	0.4235	0.5157
0.30	0.6538	0.0479	0.3908	0.5870
0.50	0.5157	0.0498	0.3744	0.7042
0.75	0.4310	0.0465	*0.3504*	*0.7724*
1.00	0.3824	0.0368	0.3327	**0.7947**

Table 4.7. Results from using only one unit per attribute. Minimum term occurence is 1 and maximum occurence is 84

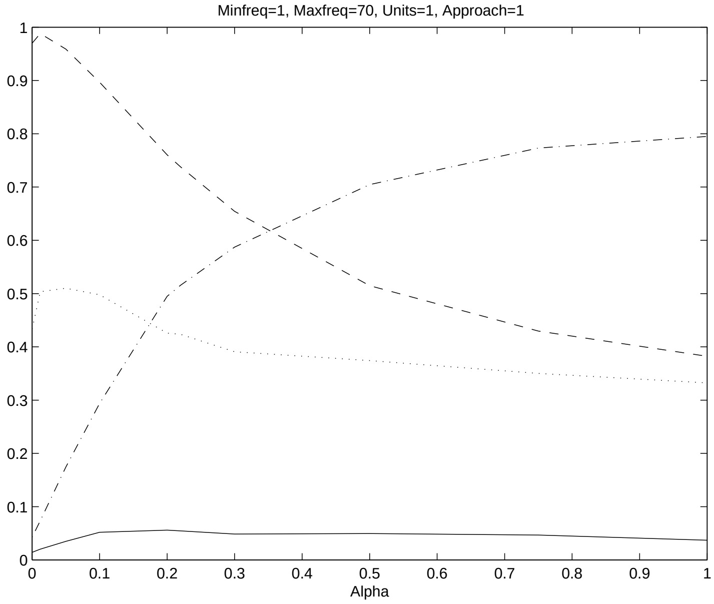

Figure 4.11. One unit per attribute, and maximum occurence 70. Measures for the plot can be found in table (4.8).

the recall is doubled, approximately 50% compared to 20%.

We are still showing higher precision values than Waern and Rudström (4.2), but a non-acceptable recall; we are loosing almost 50% of the wanted documents.

Alpha	Correct fraction	Mutual information	Precision	Recall
0.0001	0.9700	0.0144	0.4334	0.0430
1/C=0.012	0.9881	0.0202	0.5034	0.0734
0.05	0.9590	0.0349	0.5101	0.1743
0.10	0.8967	0.0519	0.4976	0.2937
0.20	0.7605	**0.0560**	0.4257	0.4948
0.22	0.7376	0.0545	0.4233	0.5157
0.30	0.6543	0.0485	0.3908	0.5872
0.50	0.5148	0.0495	0.3742	0.7046
0.75	0.4295	0.0467	*0.3501*	*0.7732*
1.00	0.3824	0.0372	0.3323	**0.7949**

Table 4.8. Results from using only one unit per attribute. Minimum term occurence is 1 and maximum occurence is 70

4.3.5 Choosing Bayesian approach estimate

One problem with the classifier is its incorrect save-predictions. This is probably due to a combination of two factors, (1) the users' frequent throw-behavior and (2) that the test-document contains many words never seen before. The system learns that the trained words in most cases are unwanted, and think that a new word is not, and makes a save-prediction.

In this section the second approach for $P(y|x_i)$ (equation (3.21)) will be tested.

Results

Conclusions

According to the results table (4.9) the precision is approximately 10% lower, and the recall much worse for the second approach than the first approach classifier from section (4.3.4). We would clearly prefer the characteristic behavior of the first approach classifier.

4.3.6 Inspecting the word weights

To reduce our suspicion of faulty behavior of the classifier we have to inspect the attributes and their weights. We pick a user who wants 50 conference calls, as we want to be sure that the weights have seen the attributes sufficient number of times.

Results

We train the classifier with all documents, and use $\alpha = 0.2$, one unit per attribute and without erasing attributes. See tables (4.10) for the 35 highest weighted at-

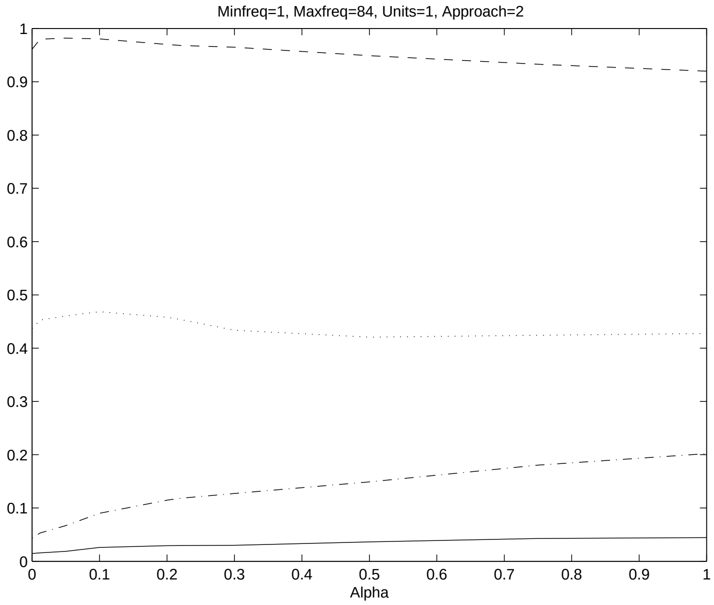

Figure 4.12. One unit per attribute, second approach. Measures for the plot can be found in table (4.9).

Alpha	Correct fraction	Mutual information	Precision	Recall
0.0001	0.9614	0.0146	0.4366	0.0437
1/C=0.012	0.9800	0.0158	0.4529	0.0532
0.05	0.9819	0.0185	0.4603	0.0668
0.10	0.9805	0.0259	0.4684	0.0899
0.20	0.9700	0.0293	0.4579	0.1146
0.22	0.9681	0.0294	0.4538	0.1182
0.30	0.9648	0.0300	0.4337	0.1271
0.50	0.9490	0.0363	0.4207	0.1489
0.75	0.9329	0.0427	0.4242	0.1804
1.00	0.9200	0.0444	0.4275	0.2020

Table 4.9. Results from using the second approach to P_{ij} when the minimum term occurence is 1 and maximum occurence is 84

tributes.

Conclusions

We clearly see that the highest weighted attributes actually have some informational meaning. They are also relevant in the area of computer science. Though, the classifiers for users who wants fewer documents are not performing equally good.

Weight	Attribute
1.25683	algorithm
1.25215	prediction
1.25215	linguistics
1.25048	philosophy
1.25048	diagnosis
1.25048	biological
1.25048	belief
1.24814	rule
1.24814	induction
1.24814	choose
1.24814	anonymous
1.24814	*vasant
1.24814	*honavar
1.24465	underlie
1.24465	ulster
1.24465	tr
1.24465	signal
1.24465	neuroscience
1.24465	nearby
1.24465	molecular
1.24465	medicine
1.24465	los_*angeles
1.24465	inquiry
1.24465	inference
1.24465	hybrid
1.24465	graphical
1.24465	graph
1.24465	grammar
1.24465	genetic
1.24465	brain
1.24465	biology
1.24465	*riichiro
1.24465	*mizoguchi
1.24465	*keane
1.24465	*informatics
1.24465	*i*j*c*a*i-99

Table 4.10. The 104 highest weighted attributes of user number 6

The wanted attributes drown in the amount of unwanted attributes. Our suspicion of faulty classfier behavior can hereby be eliminated.

4.3.7 Testing on the training set

To eliminate the possibility of that the system is working incorrectly (due to human errors while programming) we must to test on the training set. If this test fails, something is wrong.

We will test the system setup that works as good as possible, minimum occurence 1, maximum occurence 84, one unit per attribute, first approach.

Results

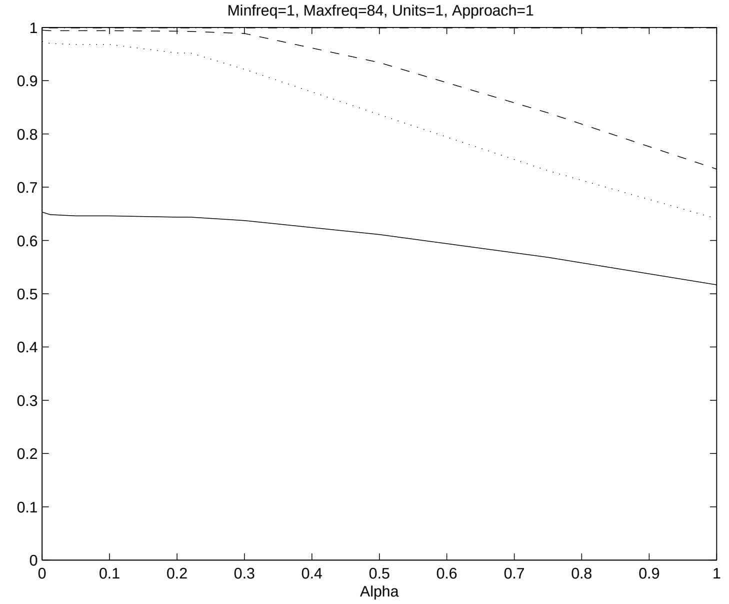

Figure 4.13. Test on the training sets. Measures for the plot can be found in table (4.11).

Conclusions

The system is behaving correctly when $0.0001 < \alpha < 0.3$. The best mutual information is approximately 0.65. The mean value of maximum mutual information over all users, $mean_{user}(I(X;Y)_{max})$ in this environment is 0.6797. The classifier has almost maximized its performance on the training set compared to this value. The bad crossvalidating behavior of the system has to origin from the nature of the problem or the model.

This test is also indicates that $\alpha > 0.3$ decreases the performance of the classifier. The precision drops as the classifier makes more and more incorrect save-predictions.

Alpha	Correct fraction	Mutual information	Precision	Recall
0.0001	0.9948	0.6533	0.9735	0.9989
1/C=0.012	0.9938	0.6485	0.9696	0.9989
0.05	0.9938	0.6461	0.9679	0.9989
0.10	0.9938	0.6461	0.9679	0.9989
0.20	0.9929	0.6438	0.9519	0.9989
0.22	0.9924	0.6437	0.9519	0.9989
0.30	0.9886	0.6374	0.9213	0.9989
0.50	0.9338	0.6110	0.8361	0.9989
0.75	0.8395	0.5682	0.7309	0.9989
1.00	0.7338	0.5167	0.6409	0.9989

Table 4.11. Results from test on the training sets. One unit per attribute, minimum term occurence is 1 and maximum occurence is 84

4.4 Using a multi-layer Bayesian Artificial Neural Network

Words in the text documents probably have relations, i.e. the words 'artificial' and 'intelligence', which destroys the probability estimates that rely on the independence assumption from section (3.2.1). This is a problem and the one-layer Bayesian Classifier, with its indepentent-assumption, may not be suitable for classification task. Introducing complex columns, here in form of word pairs may improve the classifier behavior.

How many pairs are needed? How should the pairs be chosen? This section will describe some ways to implement the Complex Columns from section (3.4), suitable or not suitable for text analysis.

We could construct all possible combinations of attributes, but all of them is probably not interesting. We could look at attribute pairs, triplets and higher orders of combinations, but we limit ourselfs to only examining pairs, due to the large memory consumption. (The matrix containing the mutual information of all possible attribute pairs is 575MB, using MATLAB. Constructing triplet-columns would make the memory requirement too big.)

We will not take into consideration whether the pairs are sequenced or unsequenced in the text document. The pairs are choosen on the basis of the mutual information between their attributes. The pair is constructed if the mutual information is higher than a set threshold.

To understand the spreading of the mutual information we look at the histogram plot (4.14).

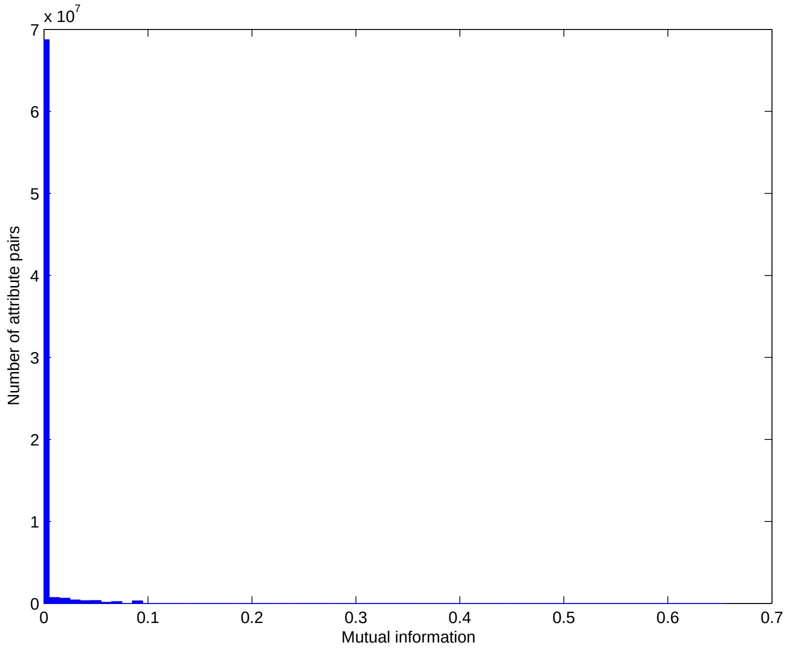

Figure 4.14. Histogram over mutual information for all attribute pairs.

We see that the mutual information is very low for almost all pairs; only 22000 of 72000000 pairs have mutual information greater than 0.1. The pairs with the highest mutual information can be found in tables (A.3), (A.3) and (A.3). The tables show the pairs with mutual information greater than 0.225, and which ones that are created when using partitioning or fragmented complex columns.

In the following tests, the first approach of the Bayesian estimate of P_{ij} (equation 3.21) and one unit per attribut will be used as they previously proved to be the best choices. We will not erase any attributes, neither low nor high frequent.

Partitioning complex columns (from section 3.4.1) and fragmented complex columns (from section 3.4.3) will be tested.

4.4.1 Partitioning complex columns

Results

As there are two interesting parameters, we choose to plot only the mutual information for all alphas over all numbers of combinations, to find the best alpha.

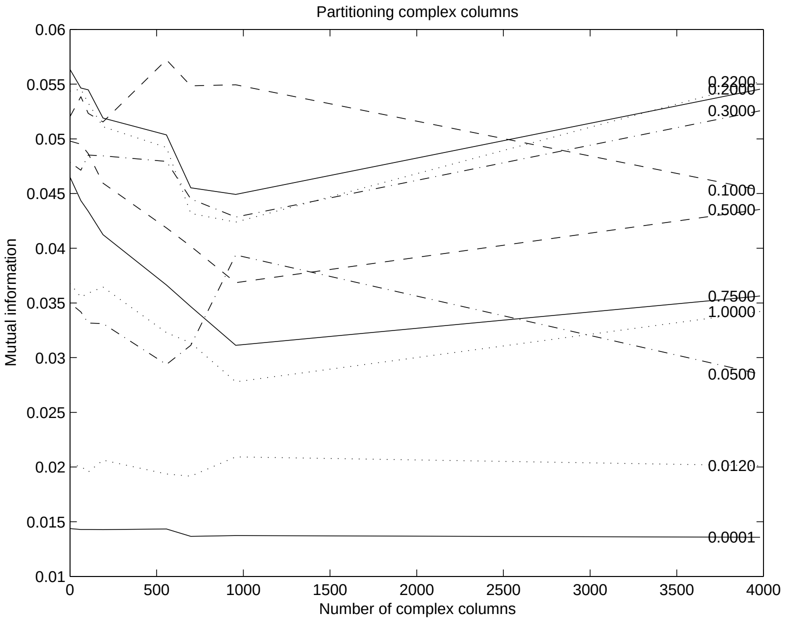

Figure 4.15. Pairs constructed with partitining complex columns. Corresponding α-values are found on the right hand side.

As seen in figure (4.15), $\alpha = 0.1$ gives the best performance, so we choose to plot all the measures for that configuration.

Conclusions

When using $\alpha = 0.012, 0.05, 0.1, 0.3$, adding complex columns increase the performance of the classifier. When using $\alpha > 0.1$, substituting 'all' single word attributes with pair complex columns seems to give as good result as not adding any combination attributes at all.

The highest mutual information can be found when using $\alpha = 0.1$ and 557 partitioning complex columns.

4.4.2 Fragmented complex columns

Here we will test fragmented columns. As we are using only one unit per attribute, fragmented and overlapping columns give the same results. (Overlapping columns creates all combinations of yes- and no-units. Here we use only yes-yes columns).

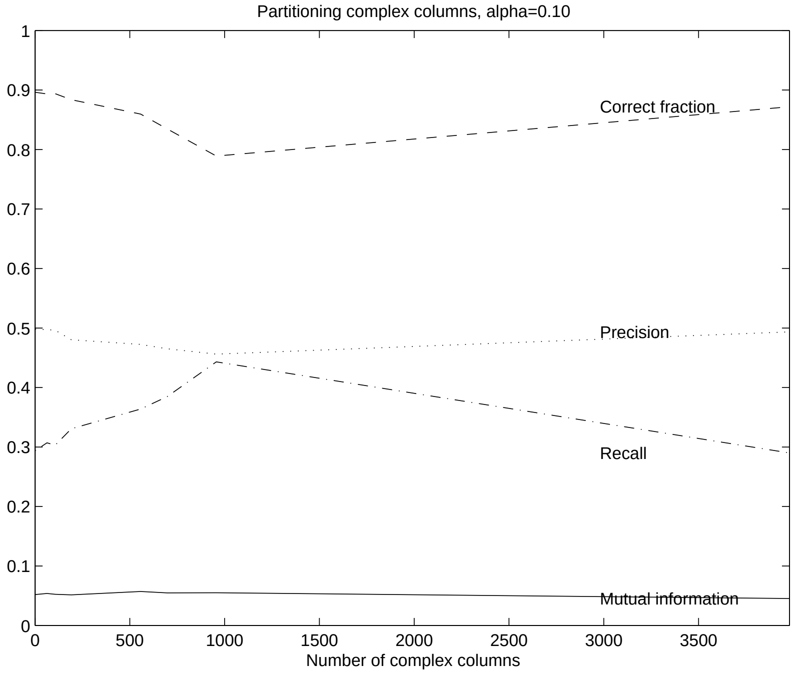

Figure 4.16. Pairs constructed with partitining complex columns. $\alpha = 0.1$. Measures for the plot can be found in table (4.12).

Number of pairs	Mutual info threshold	Correct fraction	Mutual information	Precision	Recall
0	-	0.8962	0.0520	0.4979	0.2938
63	0.225	0.8933	0.0538	0.4972	0.3069
106	0.22	0.8938	0.0523	0.4964	0.3029
191	0.18	0.8838	0.0515	0.4801	0.3307
557	0.16	0.8595	**0.0572**	0.4724	0.3639
697	0.14	0.8348	0.0548	0.4650	0.3847
956	0.12	0.7890	0.0549	0.4561	0.4432
3980	0.09	0.8719	0.0453	0.4935	0.2901

Table 4.12. Results from adding partitioning complex columns, $\alpha = 0.1$.

Results

Conclusions

Adding complex columns does not improve mutual information measure of the classifier! The more pairs we add the worse gets the performance, according to plot (4.18). The best mutual information is produced with $\alpha = 0.2$ and without any complex columns. Though, adding 150 complex columns gives almost the same

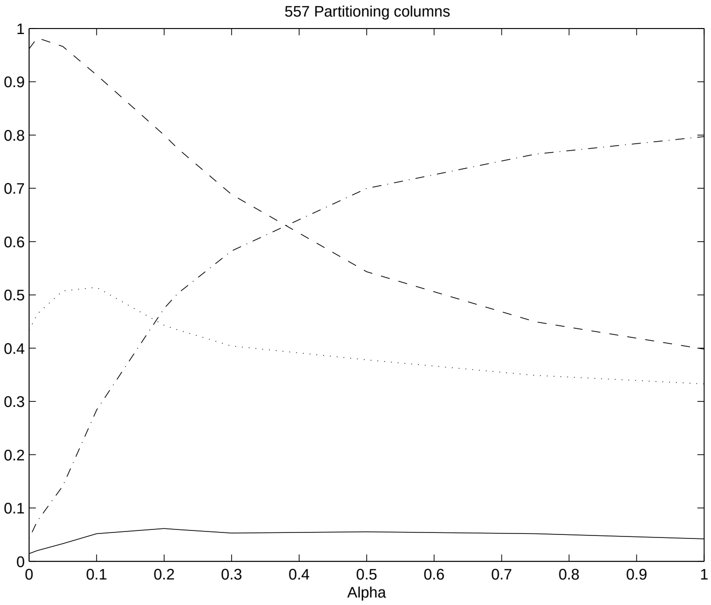

Figure 4.17. 557 partitioning complex columns with mutual information > 0.16. Measures for the plot can be found in table (4.13).

Alpha	Correct fraction	Mutual information	Precision	Recall
0.0001	0.9790	0.0143	0.4286	0.0423
0.012	0.9867	0.0194	0.5148	0.0667
0.05	0.9414	0.0294	0.4872	0.1739
0.10	0.8595	**0.0572**	0.4724	0.3639
0.20	0.7095	0.0504	0.4242	0.5242
0.22	0.6895	0.0492	0.4097	0.5484
0.30	0.5990	0.0479	0.3853	0.6008
0.50	0.4729	0.0419	0.3607	0.7083
0.75	0.3995	0.0366	0.3404	0.7719
1.00	0.3610	0.0323	0.3230	0.7943

Table 4.13. Results from adding 557 partitioning complex columns with mutual information > 0.16

performance. But if we look at the precision and recall measures both adding 150 and 328 pairs gives better results than adding non. The plot (4.19) has the same characteristics as plot (4.16) but with lower values.

Fragmented complex columns should generally require less trainingsets than prationing complex colums, but since we are using only yes-yes combinations they do

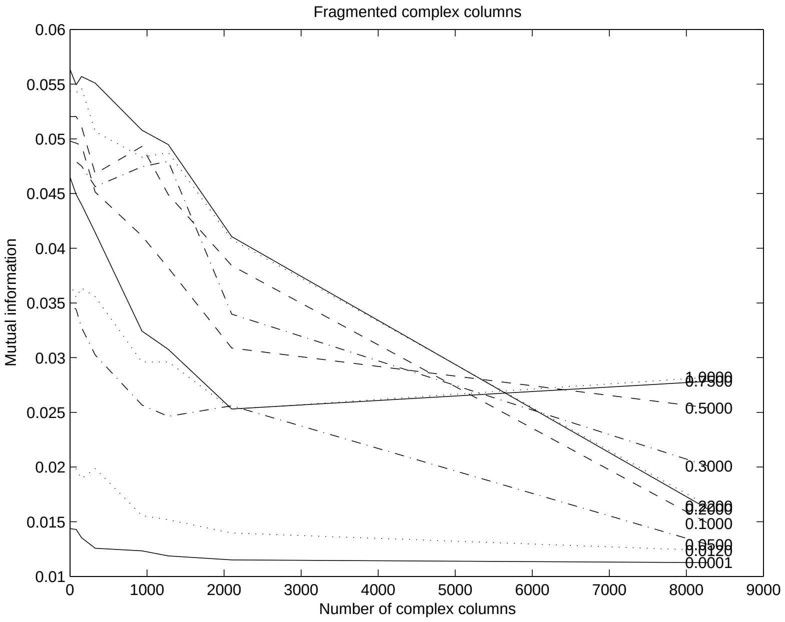

Figure 4.18. Pairs constructed with fragmented complex columns. Corresponding α-values are found on the right hand side.

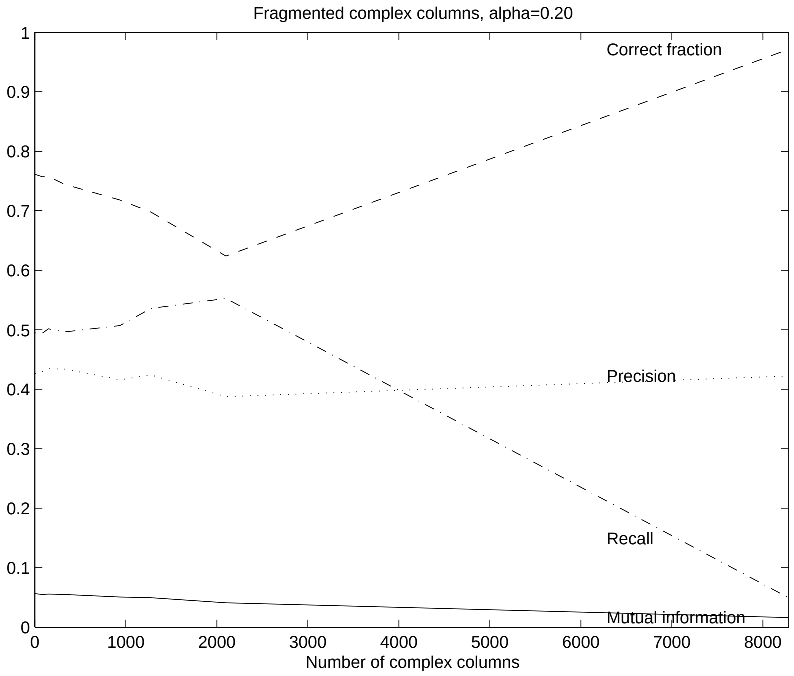

Figure 4.19. Pairs constructed with fragmented complex columns. $\alpha = 0.2$. Measures for the plot can be found in table (4.14).

Number of pairs	Mutual info threshold	Correct fraction	Mutual information	Precision	Recall
0	-	0.7614	**0.0564**	0.4258	0.4948
82	0.225	0.7571	0.0549	0.4301	0.4942
150	0.22	0.7576	0.0557	**0.4344**	**0.5016**
328	0.18	0.7443	0.0551	0.4338	0.4963
935	0.16	0.7181	0.0508	0.4160	0.5069
1277	0.14	0.6976	0.0495	0.4240	0.5358
2100	0.12	0.6238	0.0411	0.3874	0.5526
8284	0.09	0.9714	0.0161	0.4223	0.0493

Table 4.14. Results from adding fragmented complex columns, $\alpha = 0.2$.

not.

The reason why fragmented complex colums are worse than partitioning complex columns has to be the method of choosing complex columns or how the the classifier is compensated by inhibition.

4.4.3 Method of choosing pairs

In this section fragmented complex columns are used together with the partitioning method of choosing columns; a word attribute can only be used once when creating columns, even though it does not create any loops in the dependency graph. We can then exclude one of the conclusions from section (4.4.2).

If the following results show that fragmented complex columns are better than partitioning we can conclude that the method of choosing fragmented complex columns are bad. If the results show that the fragmented complex colums are worse than the partitioning we can conclude that the reason of tha bad performance is the fragmented inhibition.

Results

Conclusions

It is possible to get higher mutual information measure when adding complex columns! Several of the different alphas gets higher performance when adding columns. When $\alpha = 0.3$ and adding 3980 complex columns the mutual information measure is 0.0641, the highest value of all performed tests.

This test shows that the method of choosing how to create complex columns is crucial. Creating only one complex column including one specific attribute (partitioning method) is to prefer. Fragmented complex columns, with the inhibition of

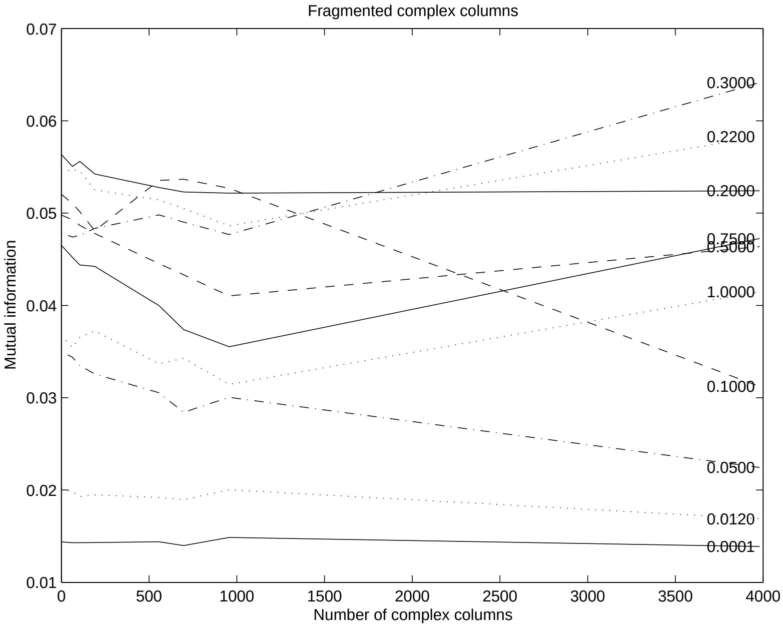

Figure 4.20. Pairs constructed with fragmented complex columns but with the partitioning method of choosing word pairs.

attributes along with the contribution of the single attribute, is the best column type.

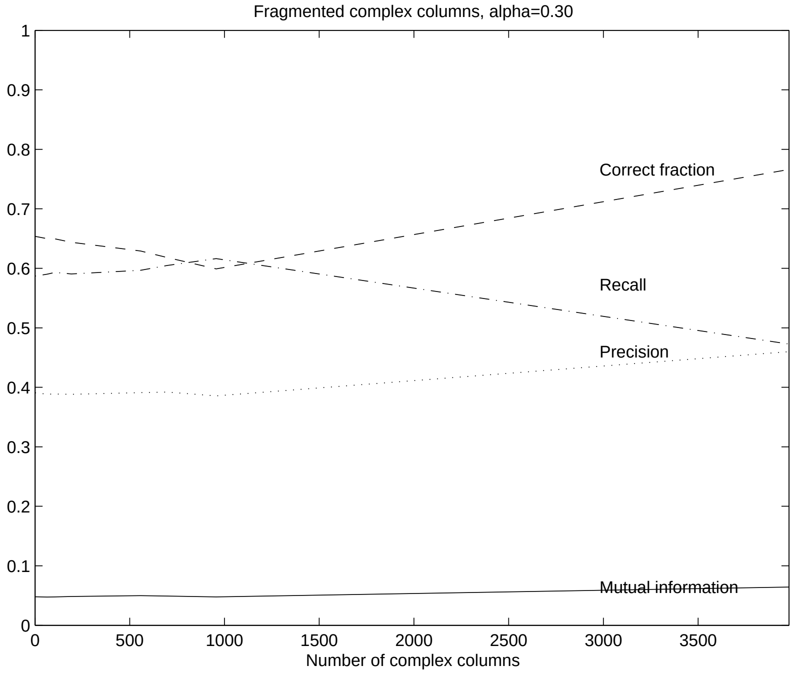

Figure 4.21. Pairs constructed with fragmented complex columns but with the partitioning method of choosing word pairs. $\alpha = 0.3$. Measures for the plot can be found in table (**??**).

Number of pairs	Mutual info threshold	Correct fraction	Mutual information	Precision	Recall
0	-	0.6538	0.0479	0.3908	0.5870
63	0.225	0.6500	0.0474	0.3884	0.5900
106	0.22	0.6495	0.0476	0.3886	0.5930
191	0.18	0.6438	0.0483	0.3881	0.5905
557	0.16	0.6290	0.0498	0.3910	0.5967
697	0.14	0.6181	0.0490	0.3919	0.6047
956	0.12	0.5990	0.0477	0.3856	0.6163
3980	0.09	0.7662	**0.0641**	0.4599	0.4726

Table 4.15. Results from adding fragmented complex columns and partitioning method of choosing columns.

Chapter 5

Discussion

In the beginning of thesis work I thought the system used would classify the data much better than it did. There are definitely a vast number of improvements to do with this method.

The best result was achieved with a one-layer Artificial Neural Network, with one unit per attribute (section (4.3.4)). Deleting low frequent attributes decreases the classifiers performance, as we are destroying the input (sections (4.3.2) and (4.3.3)). The classifier need those attributes, but it needs more trainingsets. The fact that one unit per attribute is working better than two may also indicate that the trainingset is too small. Enough information for this size of training set is lying in the yes-nodes.

Adding complex columns only made the situation worse, as they require a bigger training set (section (4.4)).

The inspected attributes in section (4.3.6) show that more wanted documents must be 'seen' by the classifier before performing properly. Also the plot (5.1), which shows the mutual information over wanted documents for the best one-layer BANN, indicates that the classifier for a user wanting more documents performs better.

Generally, for all the tests performed, a low α results in high precision and a high α in high recall.

Comparing the best result with the results from the study of Waern and Rudström [Waern and Rudström, 2000] and the reference classifier in table (5.1), we see that our classifier has a significantly better precision but worse recall.

For a task of filtering conference calls I would not pick our classifier. Its recall is too low. A low precision is not that dangerous as a low recall. Waerns classifier saves too many calls, hence the low precision. Our classifier is pickier on what to save, with the result of missing calls. Hence, the low recall value. However we

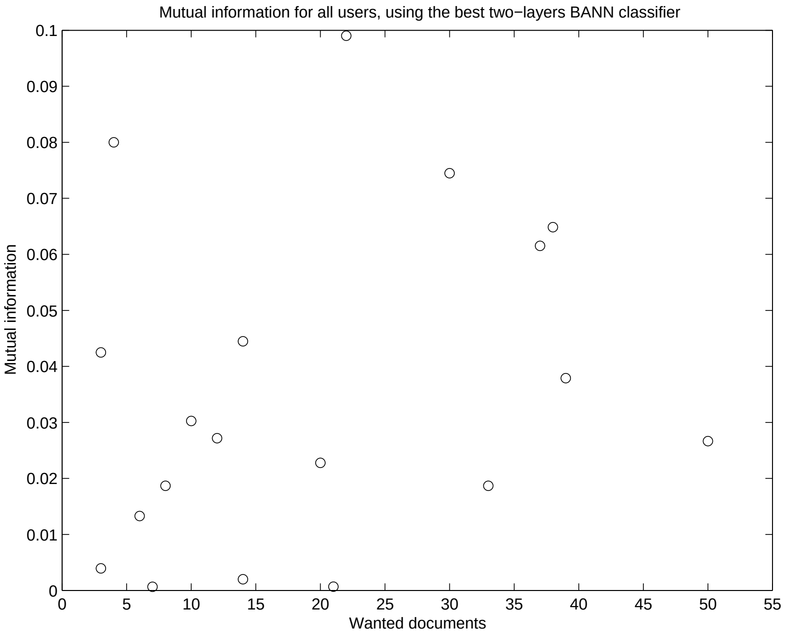

Figure 5.1. Mutual information over number of wanted documents, using the best two-layers BANN, $\alpha = 0.3$, 3980 fragmented complex columns using partitioning method of choosing columns.

can construct a classifier that is as good as the Waern and Rusdtröms modified profiles-classifier, with a one layer BANN with $\alpha = 0.75$, one unit per attribute and deleting some high occurence words.

Classifier	Correct fraction	Mutual information	Precision	Recall
Best MI, one-layer BANN	0.76	0.0564	0.43	0.49
Best MI, two-layers partitioning BANN	0.86	0.0572	0.47	0.36
Best MI, two-layers fragmented BANN	0.77	**0.0641**	0.46	0.47
As good as Waerns, one-layer BANN	0.43	0.0467	**0.35**	**0.77**
Reference one-layer BANN	0.99	0.0178	0.44	0.05
Waern Original profiles			0.21	0.70
Waern Modified profiles			0.35	0.77
Waern Generated profiles			0.31	0.62

Table 5.1. Comparison of the best classifiers, reference classifier and the classifiers from the study of Waern and Rudström. The best two-layers fragmented BANN classifier from section (4.4.3) has $\alpha = 0.3$, one unit per attribute, without erasing any attributes and using the first bayesian estimate approach.

5.1 Sources of errors

I think the biggest source of error, or reason to the bad performance, is as mentioned above, the small test set. The number of training sets are too small for compared to the large attribute set.

There is always a factor of human error in tasks like this. The system has to be constructed by hands. We have performed tests on the training set, which shows that the system behaves correctly. But of course, there might exist errors which do not show up in that kind of test.

As mentioned in section (4.1.2) the input information used was only the lemma-attribute of the word token. When doing that one loose the linguistic relations between words, and that probably results in worse predictions.

The SGML input also contains a source of error. It is not totally clean. For instance are the words 'artificial' and 'intelligence' sometimes merged into one word 'artificial_intelligence'. Furthermore should the words 'classify' and 'classifier' result in the same lemma 'class', which they do not.

What says that a word occurring several times in one document is worth less than a word occurring few times in many documents? The importance of a word does not only show in the occurence or non-occurence in a document. That measure may be to course. One could consider to calculate the term frequency of the specific word (section (2.1.1)). This requires the use of a Bayesian classifier that handles graded attribute values.

Another interesting aspect of this problem is the difference between users. Some users might not be consequent in their collection of documents. The alpha value has also been picked to suit all users as good as possible. The classifier peaks its preformance on different alphas for different users, as shown in the plots (4.8) and (4.9)

5.2 Further work

What can be done to improve the used method?

Is there any way of prefiltering the text? For example, could it be good a idea to delete all words shorter than three characters.

As tested in section (4.4), constructing more complex columns may result in better predictions. The limitation to construct complex columns representing only word pairs may be wrong. There are probably also combinations of three, four or more words that would be interesting to introduce to prevent overriding of independence assumption. The cruical point is to use a bigger train set. Constructing higher levels of combinations also requires either more processing time or memory.

Is the method used to choose pairs correct? The mutual information between input attributes gives us a value of how descriptive the word combination is to the text. But can it be considered as a good combination in a class point of view? Section (4.4.3) shows that the method of choosing pairs is cruicial. Are there other better ways to help the classifier when constructing the complex columns? Are sequencial word pairs (bigrams, trigrams etc.) better than unsequenced pairs?

It would be interesting to build complex columns representing word combinations interesting to the user. One way to determine these pairs is to use the Kullback-Leibler distance, and look at the difference in weight between the complex column and the class and the single attributes and the class.

Another improvement might be to find a method of setting the Bayesian factor α dependant of the user and his/her behavior.

5.3 My thoughts

Waern and Rudström point out the fear of loosing information when using automatic filtering [Waern and Rudström, 2000]. In their study edited buzzwords are added to the documents and the user has to create a profile containing interesting attributes. The user is also allowed to add attributes not in the buzzword list. This helps the classifier when classifying a document with new attributes.

Of course, using manually created attribute lists should perform much better in a test set as small as the one used in this thesis.

The solution of these complicated retrieval tasks is to use these mathematical approaches together with ideas from linguistics. Reducing the linguistic information to ones and zeros is crucial.

When a person investigates a document to see if it is interesting or not she usually does not read the entire text. With her knowledge about how texts is written she quickly scans a fraction of the text and makes her throw or save decision.

There is probably enough information in that section, so why must automatic filter systems read the entire text to do the task? My answer is that the approach is seriously incorrect. A classifier should, with a small in-data-window, be able to do the prediction with great result.

Appendix A

Mutual information between attributes

Here we list the attribute combinations with the highest mutual information within the document set. The listed combinations have mutual information > 0.225. A star (*) in the attribute columns means that the next character is capital. The Partitioning- and Fragmented-columns list the partitioning and fragmented complex columns which are created.

Mutual information	Attribute 1	Attribute 2	Partitioning	Fragmented
0.61270	camera	ready	x	x
0.49124	univ	univ.	x	x
0.45372	*mellon	carnegie	x	x
0.34838	data	mine	x	x
0.33879	bring	together	x	x
0.33216	agent	intelligent	x	x
0.32555	*p*a*r*c	xerox	x	x
0.32555	*honavar	*vasant	x	x
0.32555	*compaq	*s*r*c	x	x
0.32117	artificial	intelligence	x	x
0.31232	but	not	x	x
0.27954	author	must	x	x
0.27912	pa	pittsburgh	x	x
0.27912	1993	1995	x	x
0.27912	*vasant	iowa		x
0.27912	*honavar	iowa		
0.27695	reality	virtual	x	x
0.27620	p.*o	p.o	x	x
0.27620	i*r	mitre	x	x
0.27620	division	mitre		x
0.27620	division	i*r		
0.27620	clement	non-exclusive	x	x
0.27620	b*t	bt	x	x
0.27620	408	approval	x	x
0.27620	14th	arizona	x	x
0.27620	*tanaka	li	x	x
0.27620	*s*i*g*i*r	non-exclusive		x
0.27620	*s*i*g*i*r	clement		
0.27620	*mizoguchi	*riichiro	x	x
0.27620	*kobsa	alfred	x	x
0.27620	*kobe	li		x
0.27620	*kobe	*tanaka		
0.27620	*katsumi	li		x
0.27620	*katsumi	*tanaka		
0.27620	*katsumi	*kobe	x	
0.27620	*almaden	non-exclusive		x

Table A.1. Attribute pairs with mutual information > 0.225.

Mutual information	Attribute 1	Attribute 2	Partitioning	Fragmented
0.27620	*almaden	clement		
0.27620	*almaden	*s*i*g*i*r	x	
0.27384	clarity	soundness	x	x
0.26766	date	important	x	x
0.26326	no=more	than	x	x
0.26205	acceptance	notification	x	x
0.26084	discovery	knowledge	x	x
0.25867	computational	robotics	x	x
0.25811	computer	science	x	x
0.25635	high	quality	x	x
0.25602	how	participant	x	x
0.25583	l	s	x	x
0.25491	clarity	significance		x
0.25362	specification	verification	x	x
0.25158	name	title	x	x
0.24732	france	germany	x	x
0.24613	figure	table	x	x
0.24613	e	r	x	x
0.24613	c	r		x
0.24613	c	d	x	x
0.24570	p	r		x
0.24557	inc	lab	x	x
0.24557	illinois	lab		x
0.24557	c	l		x
0.24460	learn	plan	x	x
0.24460	2	3	x	x
0.23941	clarity	originality		x
0.23570	site	web	x	x
0.23549	1	2		x
0.23465	postal	reorganization	x	x
0.23380	figure	title		x
0.23322	reorganization	san_*jose		x
0.23322	northwestern	pdf	x	x
0.23322	mitre	xerox		x
0.23322	kent	ridge	x	x
0.23322	i*r	xerox		

Table A.2. Attribute pairs with mutual information > 0.225.

Mutual information	Attribute 1	Attribute 2	Pratitioning	Fragmented
0.23322	dom	visualize	x	x
0.23322	documentation	san_*jose	x	x
0.23322	division	xerox		
0.23322	convert	retrieve	x	x
0.23322	command	deployment	x	x
0.23322	china	korea	x	x
0.23322	attractive	room	x	x
0.23322	assist	work-in-progress	x	x
0.23322	arizona	documentation		x
0.23322	accessibility	disability	x	x
0.23322	14th	documentation		
0.23322	*yu	non-exclusive		x
0.23322	*yu	clement		
0.23322	*s*r*c	aggregate		x
0.23322	*s*i*g*i*r	*yu		
0.23322	*p*a*r*c	mitre		
0.23322	*p*a*r*c	i*r		
0.23322	*p*a*r*c	division		
0.23322	*compaq	aggregate		
0.23322	*bharat	as=well	x	x
0.23322	*almaden	*yu		
0.23291	leave	right	x	x
0.23118	l	p		
0.23101	proceeding	submit	x	x
0.22946	each	manuscript	x	x
0.22944	fax	tel	x	x
0.22846	discovery	mine		x
0.22841	should	than		x
0.22837	data	discovery		
0.22549	significance	soundness		
0.22537	i*b*m	talk	x	x

Table A.3. Attribute pairs with mutual information > 0.225.

Bibliography

[Billbus and Pazzini, 1996a] Billbus, D. and Pazzini, M. (1996a). A hybrid user model for news story classification. dbillbus,pazzini@ics.uci.edu.

[Billbus and Pazzini, 1996b] Billbus, D. and Pazzini, M. (1996b). Revising user profiles: The search for interesting web sites. dbillbus,pazzini@ics.uci.edu.

[Dasigi and Mann, 1995] Dasigi, V. and Mann, R. C. (1995). Neural net learning issues in classifaction of free text documents.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis.

[Dumais et al., 1988] Dumais, S. T., Furnas, G. W., and K., L. T. (1988). Using latent semantic analysis to improve access to textual information.

[EAGLES, 1995] EAGLES (1995). Evaluation of natural language processing systems, final report. Technical report, EAGLES Evaluation Working Group. EAGLES DOCUMENT EAG-EWG-PR.2.

[Fast, 1998a] Fast (1998a). *The Fast Search Engine: White Paper*. Fast Search & Transfere ASA, Brunsveien 3B, N-0667 Oslo, Norway. preliminary version.

[Fast, 1998b] Fast (1998b). *Fast SW Search: White Paper*. Fast Search & Transfere ASA, Brunsveien 3B, N-0667 Oslo, Norway. version 0.1.

[Haykin, 1998] Haykin, S. (1998). *Neural Networks: A Comprahensive Foundation*. Prentice Hall.

[Holst, 1997] Holst, A. (1997). *The Use of a Bayesian Neural Network Model for Classification Tasks*. PhD thesis, Royal Institute of Technology, S-100 44 Stockholm, Sweden.

[Kamber et al., 1997] Kamber, M., Winstone, L., Gong, W., Cheng, S., and Jiawei, H. (1997). Generalization and decision tree induction: Efficient classification in data mining.

[Keogh and Pazzani, 1999] Keogh, E. and Pazzani, M. (1999). Learning augmented bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Uncertainty 99, 7th. Int'l Workshop on AI and Statistics*, pages 225–230, Ft. Lauderdale, Florida. http://www.ics.uci.edu/~pazzani/Publications/Publications.html.

[Koller and Sahami, 1997] Koller, D. and Sahami, M. (1997). Hierachically classifying documents using very few words. In *the Fourteenth International Conference on Machine Learning, ML-97*, pages 170–178, Nashville, Tennessee.

[Lansner and Ekeberg, 1989] Lansner, A. and Ekeberg, O. (1989). A one-layer feedback, artificial neural network with a bayesian learning rule. In *Int. J. Neural Systems*, volume 1, pages 77–87.

[Lieberman, 1995] Lieberman, H. (1995). Letzia: An agent that assists web browsing.

[Mizzaro, 1996] Mizzaro, S. (1996). How many relevances in information retrieval? In *Proceedings of CoLIS2*, pages 233–250, School of Librianship, Copenhagen, Denmark.

[Murthy, 1997] Murthy, K. V. S. (1997). *On Growing Better Decision Trees from Data*. PhD thesis, The John Hopkins University, Baltimore, Maryland.

[Pazzini et al., 1996] Pazzini, M., Muramatsu, J., and Billbus, D. (1996). Identifying interesting web sites. pazzini@ics.uci.edu.

[Quinlan, 1996] Quinlan, J. R. (1996). Improved use of continous attributes in c4.5.

[Ruiz and Srinivasan, 1999] Ruiz, M. E. and Srinivasan, P. (1999). Combining machine learning and hierarchical indexing structures for text categorization.

[Salton and Buckley, 1988] Salton and Buckley (1988). Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24:513–523.

[Segal and Kephart, 1999a] Segal, R. B. and Kephart, J. O. (1999a). Dynamics of incremental learning in an e-mail classifier. rsegal,kephart@watson.ibm.com.

[Segal and Kephart, 1999b] Segal, R. B. and Kephart, J. O. (1999b). Mailcat: An intelligent assistant of organizing e-mail. In *Proceedings of the Third International Conference on Autonomous Agents*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. rsegal,kephart@watson.ibm.com.

[Waern and Rudström, 2000] Waern, A. and Rudström, A. (2000). Can readers understand their profiles? a study of human involvement in reader profiling.

TRITA-CSC-E 2013:016
ISRN-KTH/CSC/E--13/016-SE
ISSN-1653-5715

www.kth.se