Utilizing linguistic analysis in multiple source search engines

Vegard Økland

Master of Science in Informatics
Submission date: May 2011
Supervisor: Magnus Lie Hetland, IDI
Co-supervisor: Cyril Banino-Rokkones, Yahoo! Technologies Norway

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem description

Modern search engines have several data sources available to users, e.g. News search, Image search and Video search. When a user enters a query in a search engine, it is up to the user to choose a different source than the normal web search. On average, a user will only consider the first few occurrences in a search result and do so in a few seconds [36]. It would therefore be beneficial to the user experience if the user did not have to limit the sources manually to refine a search.

This project will evaluate different machine learning methods to classify relevant sources to a query. The goal of this is having an automated learning system that takes some labeled input and uses this to help inform or direct the user to the relevant source.

The project will take advantage of a Yahoo! product; *Yahoo! Query Linguist Analysis Service* (abbreviated QLAS from now on and through the document). The goal is to incorporate semantic data from QLAS into the learning system. This should augment the amount of information available to the learning system, and improve its performance. It is not clear how this semantic data could be combined with the training data and incorporated in the learning system. A substantial part of the project will be to explore this.

This project was done in cooperation with Yahoo! Technologies Norway AS (YTN). YTN develops Vespa, a search engine platform that has the possibility to search from multiple sources. YTN is interested in researching the field of learning source relevance to improve the search experience in Yahoo services. YTN is also interested in researching ways data from QLAS could be used by Vespa to enable source relevance classification when Vespa is used in a multiple-index setup.

Abstract

Modern search engines can search for multiple types of content. Average web users spend a short amount of time analyzing search engine result. These two factors make it desirable to predict what type of content are relevant for a user's query. Using this information in the search interface could lead to a better user experience with fewer user actions to access the relevant information.

This project is focused on and around the task of learning what content source is relevant for a query in a search engine. This project uses the approach of incorporating semantic data in supervised machine learning methods to achieve this. The result of the project is an analysis of a Yahoo! semantic analysis service and how it can be incorporated in a source classification system. This analysis concluded that the semantic service contain valuable information to a source classification system, but it is very expensive to utilize.

Preface

This thesis was written as the final project of my masters degree in computer science/artificial intelligence under the Department of Computer Science at The Norwegian University of Science and Technology. The project was a collaboration between the university and Yahoo! Technologies Norway.

I would like to thank my supervisors Magnus Lie Hetland and Cyril Banino-Rokkones for great feedback. I would also like to thank Yahoo! for giving me the opportunity to work on a interesting and challenging topic.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Web search engines are no longer used merely for finding textual information contained in HTML documents. The Internet has grown from a collection of hyperlinked HTML documents to a platform for *all* types of content. Because of this, search engines have adapted to search in many different types of content.

To be able to differentiate between different types of content, a search engine can maintain a separate index for each content type. This reduces the problem from deciding what *type of content* is relevant for a search, to *which source* is relevant. This simplifies the problem greatly because the classification mechanism does not need to know about the underlying differences in the content. Another benefit of this is, if one finds a good method to differentiate between some sources, it should not be difficult to extend it to a greater set of different sources.

Search engines have historically let the user decide which sources to include in a search. This is done in two ways; the user enters a term and then limits the search to a specific type of content, or the user deliberately chooses the content type by going to a specific site (e.g. `http://image.google.com`, `http://news.yahoo.com`). The average user of web search has a very short patience [36]. It would therefore be desirable to discover what content types are relevant to a query, without having the user specify it. Having the knowledge of what content types are relevant to a query could lead to an improved user experience. This should be correct considering it would require fewer actions from the user to access the relevant material.

Modern search engines will in many cases give a hint to what sources are relevant with links like “Scholarly articles for *query x*” or “Image results for *query y*” in the top of a search result. However, the methods they use are trade secrets that are not published. This makes the topic of source categorization an interesting topic.

1.1 Motivation

One way to determine which index contains the most relevant content is to send all incoming queries to all indexes and use methods from classic information retrieval to determine which result set is the most relevant (See Section 2.1 for further information). This approach will have scaling problems and be unreasonably expensive. All the indexes would have to be able to sustain the load of the biggest and most visited index. This should not be necessary. If an index is relevant and interesting to a few people (e.g Scholarly article search) it should not require the same infrastructure as a huge and highly used index (e.g. normal web search). The goal is therefore to classify the relevant source for the query, without querying any of the indexes [1].

A trivial solution to classifying relevant sources for a query could be to store each query in a dictionary together with a classified source. This could be done after some user behavior analysis have determined which source the user was satisfied with. The next time the same query is submitted to the search engine, the source classifier will classify it as the last user decided.

There are a few problems and challenges with this solution. There are no generalization of the queries, meaning that a unseen query could never be classified. According to Google, 20% of queries are unseen the past 90 days [20]. Further, the semantics of the query are not used to classify the relevant source. Another challenge is the "turn user behavior into a classification" part. While this probably is not impossible, it is not trivial either.

Using the semantic meaning of a query to determine the relevant sources should intuitively give an advantage over just ignoring it. Figure 1.1 shows a simple example where several people (or, more accurately, their respective query) have been found to have a semantic interpretation of *Celebrities* and they are labeled as relevant for the Image and Video source. Then a new person also gets the semantic interpretation of a *Celebrity* however it is not known which sources are relevant for the new person. It would then be convenient to discover relations between semantic information and sources (e.g. $semantic(Celebrity) \rightarrow relevantSource(Image, Video)$). This would enable a search engine to classify a source before the user behavior has labeled a query to be relevant for a source.

Yahoo! QLAS will be the source of the semantic information. It is not clear how the information can be utilized so this will be a significant part of this project. There are no guarantees that QLAS contains information that is relevant to source classification. Researching this and evaluating the value of QLAS in a source classification system is a big motivational factor for YTN.

[1]Possibly except an index that is dedicated to the task of classifying sources.

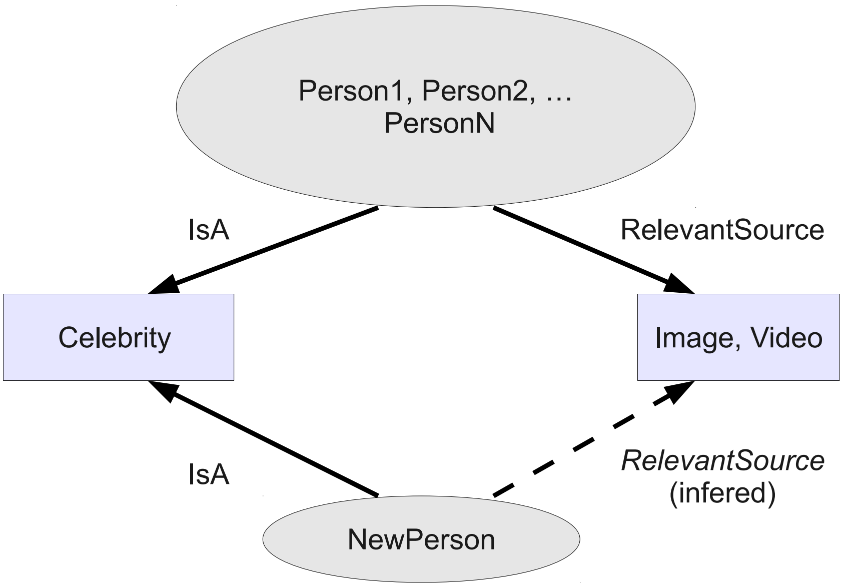


Figure 1.1: A simple example where the semantics of a query could help find the relevant sources.

1.2 Assignment interpretation

To set clear boundaries and limit the scope, the project has been split up in some independent points.

- Analyze the value of the information content in QLAS, in the context of source classification.
- Evaluate how linguistic query analysis can be used to improve machine learning methods in this field.
- Evaluate different machine learning methods in the area of classifying relevant sources from a query.

1.3 Main contribution

This report examines different ways to analyze and utilize a semantic analysis of a search query to classify what source are relevant for the query. The report examines traditional feature vector machine learning, parameter free machine learning, metric indexing and using traditional information retrieval to achieve this.

The report gives a detailed analysis of the value and usability of a specific semantic analysis tool available at Yahoo!.

Some experimental software was written to test these mechanisms and some 3rd-party libraries and software packages were used.

1.4 Report outline

Chapter 2, *Background theory and state of the art*, gives an introduction to classic information retrieval, machine learning, metric indexing and the Yahoo! service QLAS, that this project utilizes extensively.

Chapter 3, *Experiments*, explains the experiments that make the base of the project. This chapter is heavily based on the theory given in Chapter 2.

Chapter 5, *Results*, states the results of all the experiments given in Chapter 3. It also gives a simple comparison of the results in the different experiments.

Chapter 6, *Result analysis and comparison*, gives a deeper analysis and comparison of the results in Chapter 5.

Chapter 7, *Conclusion and future work*, summarized the results of the project and suggest areas of further work.

Appendix A, *APPENDIX*, contains example documents and outputs from QLAS, QLAS documentation and the full output from the machine learning experiments.

Chapter 2

Background theory and state of the art

This chapter gives the necessary background knowledge about machine learning, traditional information retrieval methods, metric indexing and software packages used and referenced in the project.

2.1 Traditional Information retrieval

The field of information retrieval has traditionally been focused on making a large collection of text documents readily available. A second objective is to rank the relevance of the different documents in regard to a user query. This field is much older than computer science, libraries have used card catalogs for indexing and book classification systems like the Dewey Decimal Classification and Library of Congress Classification to discriminate relevance between different books. The goal of these library systems are much the same as in modern information retrieval, to navigate a large document collection where it is unfeasible of a user to do a sequential scan to find the desired content.

In the field of artificial intelligence, the problem of information retrieval is seen as a classification problem [31], where each document can be divided into two classes; *relevant* and *non-relevant*. To achieve this a model is built around the documents and different models facilitate different information retrieval methods. Examples of these models are the *Boolean Model* which uses Bayesian methods (e.g. Naive Bayes) and the *Vector Model* which uses methods like *term frequency–inverse document frequency* and *cosine distance.*

Several information retrieval techniques have been developed over the years. This subsection will give a short introduction to the different techniques.

2.1.1 Document indexing

The purpose of creating a document index is to create a data structure that enables a fast look up through a large document collection. This should be possible without having to do a linear scan (sequentially reading through each document) of all the original documents.

Inverted index

An inverted index is a data structure that maps each individual piece of content (e.g. word, word span, phrase) to the location where the content is stored [4]. An example of an inverted index can be seen in Table 2.1. The index consists of three documents:

Doc1: Oranges and lemons, Say the bells of St. Clement's.

Doc2: You owe me five farthings, Say the bells of St. Martin's

Doc3: When will you pay me? Say the bells of Old Bailey.

Content	Document	Content	Document
and	1	owe	2
bells	1 ,2 ,3	pay	3
farthings	2	say	1, 2, 3
five	2	st. clement	1
lemons	1	st. martin	2
me	2, 3	the	1, 2, 3
of	1, 2, 3	when	3
old bailey	3	will	3
oranges	1	you	2, 3

Table 2.1: A small inverted index . Some pre processing have been done on the data.

Vector space model

The vector space model is a method of representing a document as a vector (or a document collection as a matrix) where each position in the vector represents a term [4].

Using the same three documents as in Section 2.1.1 the matrix would look like Table 2.2

Doc	and	st.martins	old	st.clements	owe	of	farthings	me
1	1	0	0	1	0	1	1	0
2	0	1	0	0	1	1	0	1
3	0	0	1	0	0	1	0	1

...

Doc	say	bailey	five	oranges	the	lemons	you	bells
1	1	0	0	1	1	1	1	1
2	1	0	1	0	1	0	1	1
3	1	1	0	0	1	0	0	1

Table 2.2: Example vector space model

2.1.2 Relevance evaluation

The vector space model, in combination with an inverted index (or other index) enables one to do a fast evaluation of relevance. The relevance can be calculated by using a combination of *term frequency–inverse document frequency* (*tf-idf*) and *cosine similarity* to calculate the distance of two documents [32].

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2.1}$$

$$\mathrm{idf_i} = \log \frac{|D|}{|\{d : t_i \in d\}|} \tag{2.2}$$

$$(\text{tf-idf})_{\mathrm{i,j}} = \mathrm{tf_{i,j}} \times \mathrm{idf_i} \tag{2.3}$$

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{2.4}$$

The tf-idf function (Equation 2.3) is used to calculate the importance of each term in a document by giving common terms a low score and more unique terms a higher score. An example of this is labeling the term "*the*"[1] as unimportant for an English document and some unique terms as highly important.

The cosine similarity measure (Equation 2.4) is then used to calculate the cosine of the angle between two documents. A small angle gives a greater similarity in the range $[0, 1]$ than a bigger angle.

[1]Common terms like "the" are often removed because they convey very little information, but one does not have to do so. This process is called stop word removal.

2.2 Machine Learning

Deciding which sources are relevant to a query can be seen as a *multiclass statistical classification* problem. Statistical classification is a sub field of supervised machine learning. Another example of this type of classification is *language detection* for natural language documents.

There exists several statistical classification methods and the various methods perform differently on different problems and data sets. There is no clear better method for all problem domains [38, 37, 39]. One should therefore evaluate several methods to find the one appropriate for the given problem. This notion of there being no single best method is often referred to as the *no free lunch theorem*.

2.2.1 Supervised versus unsupervised machine learning

One can separate a classification task into two families. If the data is in the form $example_m \rightarrow class_x, example_n \rightarrow class_y$ the method is *supervised*. If the data is in the form $example_m, example_n$, without class information, the method is *unsupervised*.

The difference in supervised and unsupervised learning is the type of training data it uses. Supervised learning uses training data that is labeled with a correct class or "answer". The data is usually in the form of a pair $(input, output)$. Unsupervised learning uses data without output examples. Both methods are important techniques. It can be difficult and/or costly to obtain classified training data, hence it would be a great limitation to only be able to do supervised learning.

If one is using unsupervised learning to solve a classification task, the task is called *clustering*. The method can not answer a question as "what language is this document" but it can do a task "find documents with a similar language like this one".

2.2.2 Eager vs lazy machine learning and inductive bias

At the root of all machine learning is an *inductive bias*. This is the formal way a certain method generalizes from seen instances in the training data to unseen instances. Without an inductive bias, a machine learning method is nothing more than a dictionary.

An eager machine learning method will make this generalization before any unknown instances are seen, e.g. by creating a decision tree from the training data. The inductive bias of this example will be in what fashion the decision tree is created and how it is discriminated from other alternative trees. ID3 [27] does, for example, prefer small trees over large ones and uses a entropy based function *information gain* as a heuristic to do so.

A lazy method postpones the inductive bias until an unseen instances is presented. The entire training data corpus is used as a possible source of classification and the inductive bias is usually in how cases from the training corpus are selected as sources of classification. Examples of lazy methods is *k-nearest-neighbor* (*kNN*) and *case based reasoning*. kNN works by choosing the k training cases that lie closest to the unclassified case, using some distance function. Here the inductive bias is in the choice of distance function, the size of k and how the k closest objects are treated as sources for classifying the unknown instance. There are several ways to use the k closest objects to classify an object, e.g. using them as votes or using the distance to each object. A more advanced method is placing the k objects in a semantic ontology and using a form of semantic classification.

2.2.3 Parameter free machine learning

A machine learning method that has many tunable parameters can be problematic. The person using the method will have to use his own prejudice to tune the parameters. This will result in a human influence on the result from the machine learning. Keogh et al. [22] deals with the problem of parameter-laden data mining algorithm and promotes parameter-free algorithms. From the abstract of the paper:

> Data mining algorithms should have as few parameters as possible, ideally none. A parameter-free algorithm would limit our ability to impose our prejudices, expectations, and presumptions on the problem at hand, and would let the data itself speak to us.

In this paper they highlight that algorithms with many parameters are difficult to compare. They compared 51 algorithms to their approach, inspired by Li et al. [23]. They tested all the algorithms by clustering eighteen pairs of time series data from different sources. In comparing the algorithms they discovered that while the parameter-laden methods could be tweaked to perform well on all the data sets, for ¾ there was no single set of settings that performed well over all the data sets.

The result from this paper speaks strongly in favor of parameter free algorithms. The paper explored the parameter free function *Normalized compression distance*. This algorithm is presented in Section 3.2.2.

2.2.4 Challenges and sources of error in machine learning

A machine learning algorithm can run into challenges because the training data is not perfect or the appropriate amount of training data is unknown.

Noise

Noise in training data can be contradicting information or random data that is not relevant to the target model. This imposes challenges on the machine learning algorithm and different algorithms tackle this in different ways. (The next section contains more about this.)

Overfitting

Overfitting in machine learning is when a model is over trained or under trained [18]. This can happen when one use too much or not enough training data or training time. The goal of the training model and its inductive bias is to learn generalizations of a model domain from the training data. If overfitting occurs it learns features or random noise that are specific to the training data which is not representable for the general model domain. Another way to state this problem is if the trained model violates Occam's razor or the principle of parsimony by using an overly complex model, e.g. by using a quadratic function to model a linear model.

There are several methods to prevent overfitting, some are listed below.

- **Cross validation** can be done in many ways. The general idea is to partition the training data into two complementary partitions, one for training and one for analysis. After a model is created from the training set the analysis partition is used to measure how well the model performs. This analysis can be done several times and the result from it can be used in several ways, e.g. stopping the training early, send a feedback that more training is required or used as a heuristic in pruning the trained model.
- **Bootstrap aggregating** or **bagging** [7] works by creating several subsets from a set of training data by using uniform sampling with replacement. This creates several sets where many of the samples will be represented in more than one set. Each of the sets are used to create their own models using some machine learning algorithm. Each of these models are then combined to do classification by averaging the result of each, or use each model as a vote. The result of this is a smoothing effect on the final model.
- **Pruning** is a method to reduce the size of decision trees [17]. The optimal size of a decision tree is difficult to find. If it is too small it will neglect important

features in a model. If it is too large it risks overfitting the data. Pruning a tree is to remove parts of it, if doing so does not reduce the performance of the model. Cross validation can be used to measure this.

2.2.5 Entropy based classification

Benedetto et al. introduced the idea of using a common compression algorithm like LZ77 to measure the relative entropy between two pieces of data. This entropy measurement can be used to classify data of an unknown class.

The entropy of a piece of data is a measurement of how "predictable" or uncertain a random variable is. Entropy usually refers to *Shannon entropy* [35]. This is the measurement of the *average missing information content* if one does not know the value of a random variable. Shannon entropy also defines the theoretical limit of an optimal loss-less compression of a piece of data. If a data source is an infinite proper/true random source, its entropy is the same size as the source. I.e. it is impossible to predict the value of random variable and the data can not be compressed. If a source is more predictable, e.g. a natural language, it has a finite entropy which is smaller than the original source. This stems from the fact that certain characters or words are used more frequently than others, e.g 'a' and 'e' are used more than 'z', and 'the' is used more often than 'xylophone'.

A different definition of entropy, which suites this context well is the *Chaitin–Kolmogorov* complexity: the complexity of a string is the length of the string's shortest possible description in some fixed universal description language. Unfortunately, it is computationally impossible to find this description [23]. It is however upper semi-computable, meaning it can be approximated from above. Common compression algorithms try to achieve this approximation and it is therefore a powerful tool to measure entropy.

There are appealing properties in using the approximated Kolmogorov complexity to do classification. It requires no a priori knowledge about the data, it has no requirements considering data format (e.g. feature vector) and it is *parameter free* (see Section 2.2.3). Because of this, in addition to being a lazy learning method, makes it a very slow method and might not be practical to use in a real world system. In the end it serves as an interesting way to analyze data. It is also interesting to compare it against other methods that either require a priori knowledge or have requirements regarding the training data format.

Using entropy without compression

There are other ways of measuring entropy of data and machine learning algorithms have done so for a long time before researchers started experimenting with compression and classification. The ID3 algorithm [27], published by Quinlan in

1986 uses entropy to measure *information gain.* This is used as a heuristic to create small decision trees. In contrast to using data compression, ID3 expects its training data to be a feature vector. The entropy of each feature in a feature vector is measured individually to find which feature that splits a data set in the best way. It does not evaluate the entropy of the entire data set as a whole.

2.2.6 Statistical classification methods

This subsection gives a short summary of six statistical classification methods. Each of them are refered to in Section 3.3.

Naive Bayes classifier

Naive Bayes is a statistical classifier based on Bayes theorem and the assumption that all variables are statistically independent [31]. A Naive Bayes classifier can be stated as Equation 2.5 and rewritten as Equation 2.6. Equation 2.6 clearly shows the independence assumption. The probability of a class is the product of the probability of the class and the probability of each feature. This independence assumption is not necessarily correct but the algorithm may still perform well [40].

$$p(C|F_1, ..., F_n) = \frac{p(C)p(F_1, ..., F_n|C)}{p(F_1, ..., F_n} \tag{2.5}$$

$$p(C|F_1, \dots, F_n) = \frac{1}{Z} p(C) \prod_{i=1}^{n} p(F_i|C) \tag{2.6}$$

Decision tables + Naive Bayes

This algorithm combines a Naive Bayes model and a decision table to create a *semi-naive Bayes classifier* [15]. The decision table represents a Bayesian network that describes conditional dependencies between variables. The goal of the decision table is to find highly discriminating variables. This is done by selecting variables that maximizes cross-validating performance. A class probability is given by combining the Naive Bayes and the Decision Table. This can be seen in Equation 2.7. $p_{NB}(C|X^{\perp})$ is the probability given by Naive Bayes, $p_{DT}(C|X^{\top})$ is the probability given by the Decision Table and α is a tuning constant.

$$p(C|X) = \frac{\alpha \cdot p_{DT}(C|X^{\top}) \cdot p_{NB}(C|X^{\perp})}{p(C)} \tag{2.7}$$

IB1

IB1 is a *instance based learner* which means that the inductive bias is postponed until a test case is presented. IB1 uses Equation 2.8 to calculate the distance between the test case and training instances. The closest class is selected as the class of the unknown test case [3].

$$\begin{aligned} Similarity(x,y) &= -\sqrt{\sum_{i=1}^{n} f(x_i,y_i)} \\ f(x_i,y_i) &= (x_i - y_i)^2 \end{aligned} \tag{2.8}$$

RIPPER

Repeated Incremental Pruning to Produce Error Reduction (RIPPER), was presented by Cohen [11] and is a optimization of IREP (Incremental Reduced Error Pruning [14]). RIPPER is a prepositional rule learner that tries to learn first order logic (FOL) rules efficiently on big and noisy datasets. Rules are created by splitting the training data in two sets; a growing set ($^2/_3$) and a pruning set ($^1/_3$). The rules are created by first creating a single rule that covers the growing set, the pruning set is then used to delete conditions that maximizes a cross validation function. Instances that are covered by the rule is then deleted and the rule is added to the rule set. This process is repeated until all the training instances are covered by a rule.

C4.5

C4.5 [28] is a decision tree algorithm and is an extension of ID3 [27]. C4.5 builds a decision tree in the same way as ID3, by using information entropy. The improvements over ID3 are listed below.

- C4.5 handles continuous attributes. It does this by creating thresholds at nodes that splits the lists by dividing the list into those variables above and below the threshold.
- C4.5 handles missing values. It does this by marking missing values as **?** and those values are not used to calculate information gain.
- C4.5 prunes the tree after creation. This is done by going through the tree after it is created and collapsing branches into leaf nodes if the branch does not improve the cross-validating performance.
- C4.5 handles attributes with differing costs.

Random Forest

A random forest classifier creates a forest of decision trees [8]. During classification the random forest evaluates the unknown instance against each decision tree in the forest and selects the *mode* (the class that has the largest number of decision trees). Each decision tree is created by m random attributes and splits are created by how well the attribute splits the training set.

2.2.7 Training data

Different machine learning methods require different types of training data. This section explains *feature vectors*, *semantic networks* and *entropy data.* The most common of these are feature vectors, used by algorithms like *ID3*, *C4.5*, *Naive Bayes*, and *random forest.* Semantic networks are used in e.g. *case based reasoning* [2].

There is one big challenge when using complex training data and the algorithms you use require feature vectors or semantic networks: How one transforms the original data (e.g. medical images, time series, audio clips, video clips) into another form that is more suitable for the machine learning algorithm? An even greater challenge is to do that *without loosing any information* contained in the original data.

Feature vectors A training set of feature vectors often look like the example in Table 2.3 and the cases with an unknown classification/answer exemplified in Table 2.4. Each feature can be binary or n-ary attributes, strings or floating-point values. Each algorithm using feature vectors will have a specification about what types of attributes it can take as valid input.

$feature_{a1}$	$feature_{b1}$	$feature_{c1}$	...	classification
$feature_{a2}$	$feature_{b2}$	$feature_{c2}$	...	classification
$feature_{a3}$	$feature_{b3}$	$feature_{c3}$	...	classification

Table 2.3: Feature vector training data

$feature_{a1}$	$feature_{b1}$	$feature_{c1}$	...	???
$feature_{a2}$	$feature_{b2}$	$feature_{c2}$	...	???

Table 2.4: Feature vector to be predicted

Semantic networks Semantic networks are structured data surrounding some other piece of data in a given domain. Methods that use this will typically use the semantic information to do more complex similarity measures. An example of one

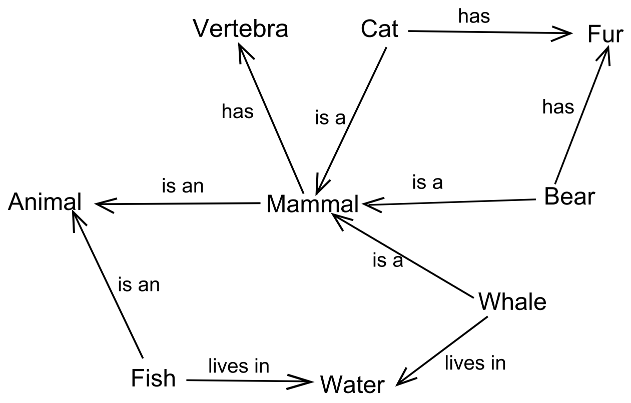


Figure 2.1: A semantic network from the animal kingdom

of these methods is *case-based reasoning*. Figure 2.1 is an example of a semantic network.

Entropy data All data have a Kolmogorov complexity which is a measure of how much information is contained in the data. The Kolmogorov complexity is the theoretical measure of how much data can be compressed. This can be used in entropy based machine learning. All data have some entropy and there are no requirements on the form or size of the data. See Section 2.2.5 for more information about Kolmogorov complexities.

2.2.8 Training data in this project

The training data for this project is a feature vector model supplied by Yahoo! Taiwan, see Table 2.5 for an example. It consists of a query, three probabilities and some identification variables. The data set is split up in 15 164 test cases and 138 699 training cases evenly divided in four classes. The data were not used "as is" in the experiments because the focus of using QLAS in the classification process. The important part of the original training data in this project is the query, its classification and its QLAS analysis.

The linguistic analysis from QLAS will be used to improve the machine learning process. Exactly how this service should be used is not obvious and a major part of this project is to explore this problem. A further description of the linguistic service can be found in Section 2.4.1 and an example of a analyzed query can be found in Section A.1.1.

NO	JUDGMENT	SPVID	QUERY	slm.web	slm.image	slm.shopping
0	0.0	.AP3eg837...	中翻英	-4.624399	-19.55382	-6.811815
0	0.0	.Gh9yne...	sd暴力美學	-9.99194	-26.07177	-11.30722
	....					

Table 2.5: Training data example. The numbers in the right three columns are log-likelihoods

2.3 Metric Indexing

Metric indexing is a type of similarity search that can be used where traditional (coordinate-based) spatial access falls short. This is typically the case when one is searching in collections of complex documents, e.g. multimedia files or scientific data. When using metric indexing to facilitate search, a query is an object equal in format to those in the document collection. An example of this is how an image would be the query in a image search engine.

Metric indexing search is based on a *distance function* and the goal of metric indexing is to do some form of pre processing to achieve better performance than a *linear scan* (applying the distance function on the query object and each of the objects in the collection). This distance function must adhere to the mathematical notion of a *metric*, hence metric indexing [19]. A distance function $d : X \times X \rightarrow R$ must adhere to the following constraints to qualify as a metric:

1. Non-negative, $d(x, y) \geq 0$
2. Symmetric, $d(x, y) = d(y, x)$
3. Triangle inequality, $d(x, z) \leq d(x, y) + d(y, z)$
4. Identity of indiscernibles, $d(x, y) = 0$ iff x and y are equal.

2.3.1 Search types

A *linear scan* is the most basic way to do a similarity search. To improve this one can calculate or estimate $d(query, object)$ without actually calculating the distance function. This can be done by taking advantage of the metric space, as long as the approximation adheres to some restrictions; the approximation must *always* be *either* an under estimate or an over estimate. These functions are denoted $\hat{d}(x, y) \geq d(x, y)$ and $\check{d}(x, y) \leq d(x, y)$.

The most basic type of search when using metric indexing is a *range search*. This search finds objects that lie not further than *range* from the query object, $d(query, object) \leq range$. By using the functions $\hat{d}$ and $\check{d}$ one can make some statements about the relevance of an object, summarized in the following table.

The task of excluding objects from a collection is more important than including the relevant. This is usually correct because the majority of a document collection is not relevant to a query.

Distance	Relevancy
$\hat{d}(query, object) > range$	maybe relevant
$\hat{d}(query, object) < range$	relevant
$\check{d}(query, object) > range$	not relevant
$\check{d}(query, object) < range$	maybe relevant

Another search that have many practical applications is *k-nearest neighbors (kNN)*. This can be implemented using the range search. The search is started by setting the radius to infinite and until k objects are found the radius stays infinite. After k objects are found the radius is decreased using the k objects as a heuristic. If an evaluated object lies outside the current radius it is discarded. If it is within the radius it is added to the candidate set and the radius is possibly decreased. This type of search is a *branch and bound* method.

2.3.2 Curse of dimensionality

Different data sets and distance metrics have a different dimensionality. When the dimensionality of data increases it becomes more difficult to index. This is often called the *curse of dimensionality*, a term coined by Richard E. Bellman [5]. The reason for this curse is the exponential growth of space by linearly adding dimensions.

To measure the dimensionality in regards to metric indexing one can use *intrinsic dimensionality* [10] defined in Equation 2.9. The variables μ and σ are the mean and standard deviation of the *distance distribution histogram* of the data set. An example of a distance distribution histogram can be found in Figure 6.2.

$$\rho = \frac{\mu^2}{2\sigma^2} \tag{2.9}$$

The intrinsic dimensionality is high if the distances between objects are very similar. The extreme case is a "zero-one metric", where the distance between all objects except itself is zero. The intrinsic dimensionality increases towards infinity as the zero-one index grows.

2.3.3 Index data structures

There are several index types one can use in metric indexing. This section presents some common concepts and three specific index structures.

Pivot based index structures

Pivoting is a method where some objects, either a subset, the full set of the object collection, or a synthetically generated set, are chosen as *pivots*. The selection of pivots can either be at random or using some heuristic, e.g choosing objects that are distant from each other. The distance from each object to each pivot is calculated and stored in a matrix. This is the index structure used when searching.

To be able to utilize the index one have to be able to estimate $\hat{d}(q,x)$ and $\check{d}(q,x)$ without calculating $d(q,x)$. This can be done by using the geometry of the metric space. $\hat{d}(q,x)$ is set to $\hat{d}(q,x) = d(q,p) + d(p,x)$ and $\check{d}(q,x)$ is set to $\check{d}(q,x) = |d(q,p) - d(p,x)|$.

When a query is evaluated the distance $d(query, pivot)$ is calculated for each pivot. This distance and the distance $d(pivot, object)$ taken from the index is then the data needed to calculate the upper bound. To take advantage of multiple pivot objects in the pivot set P the functions $\hat{d}(q,x)$ and $\check{d}(q,x)$ are defined as $\hat{d}(q,x) = \min_{p \in P} \hat{d}(q,x)$ and $\check{d}(q,x) = \max_{p \in P} \check{d}(q,x)$. Figure 2.3 shows a graphic representation of how the search space is reduced by evaluating several pivots.

AESA Approximating and Eliminating Search Algorithm (AESA) [30] uses all the objects in a data set as pivots. At the start of a query, the distance from the query to a random pivot is calculated. This distance is used to calculate the lower bound $\check{d}$ (and possibly the upper bound $\hat{d}$) given in Section 2.3.3. After this initial distance calculation a filtering is performed removing all objects that have $\check{d} > r$. If one finds objects with $\hat{d} < r$ it can be included in the result set without calculating the distance. A new pivot is then chosen and the procedure is repeated.

This method has a high memory and creating cost because of the choice to include *all* objects as pivots, $O(n^2)$.

LAESA Linear AESA [25] address the high CPU and memory cost of AESA. It makes a compromise and only uses a subset of the objects in the data set as pivots. This gives a cost of $O(nm)$, where m is the size of the pivot set. A LAESA index has the same form as AESA if $n = m$, but the search procedure is different. Because it uses fewer objects as pivots it can not be expected to filter as well AESA. This will lead to more distance calculations when performing a search.

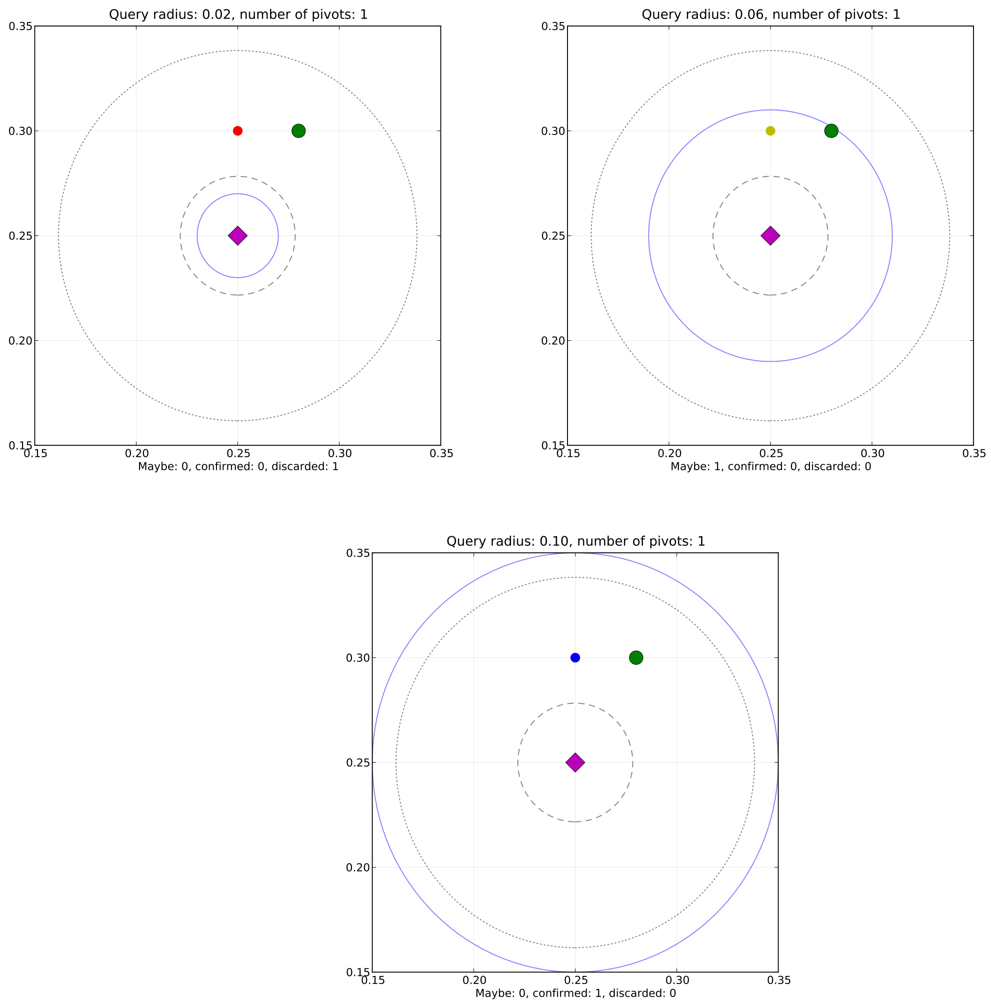


Figure 2.2: The figure shows three radius values and the distance approximation functions $\hat{d}(q,x) = d(q,p) + d(p,x)$ (outer dotted line) and $\check{d}(q,x) = |d(q,p) - d(p,x)|$ (inner dashed line). The query object is the purple object in the center, the pivot object is green and the unknown object changes color. Using three different radii, one can discard (first plot) or confirm the object (last plot), or the state could be unknown (second plot). If the state is unknown one has to calculate the distance function $d(query, unknown)$. As stated earlier, the goal is to minimize the number of these calculations.

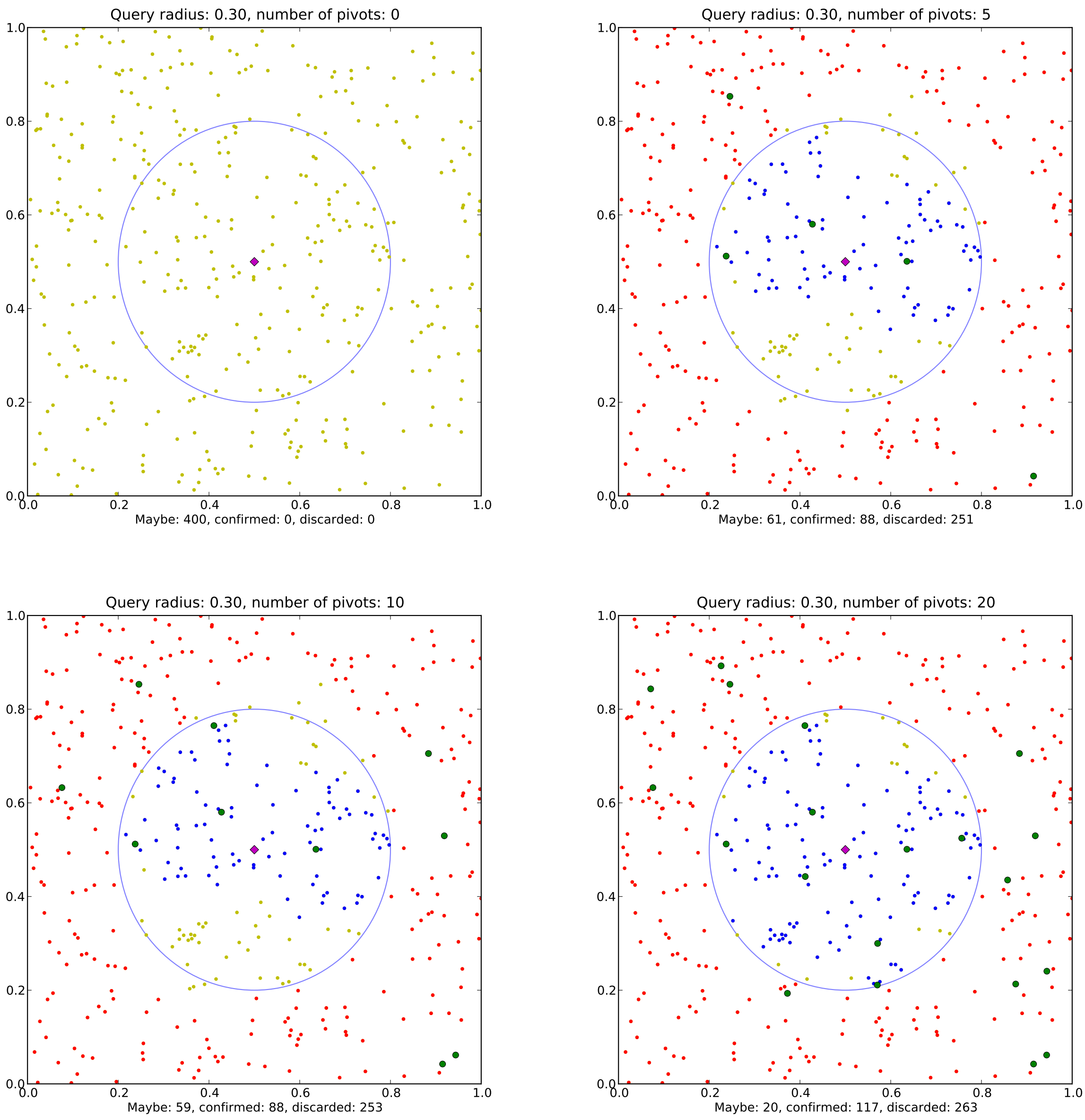


Figure 2.3: The figure is showing how the search space is reduced at each step of considering multiple pivots, using the functions $\hat{d}$ and $\check{d}$ from Section 2.3.3. Blue objects are definitely relevant, yellow are possible matches and red are discarded objects. By calculating the distance to 20 pivot objects the search space are reduced from 400 objects to 117 confirmed, 263 discarded and 20 unknown. After the filtering is done one have to calculate the distance to the 20 unknown objects .

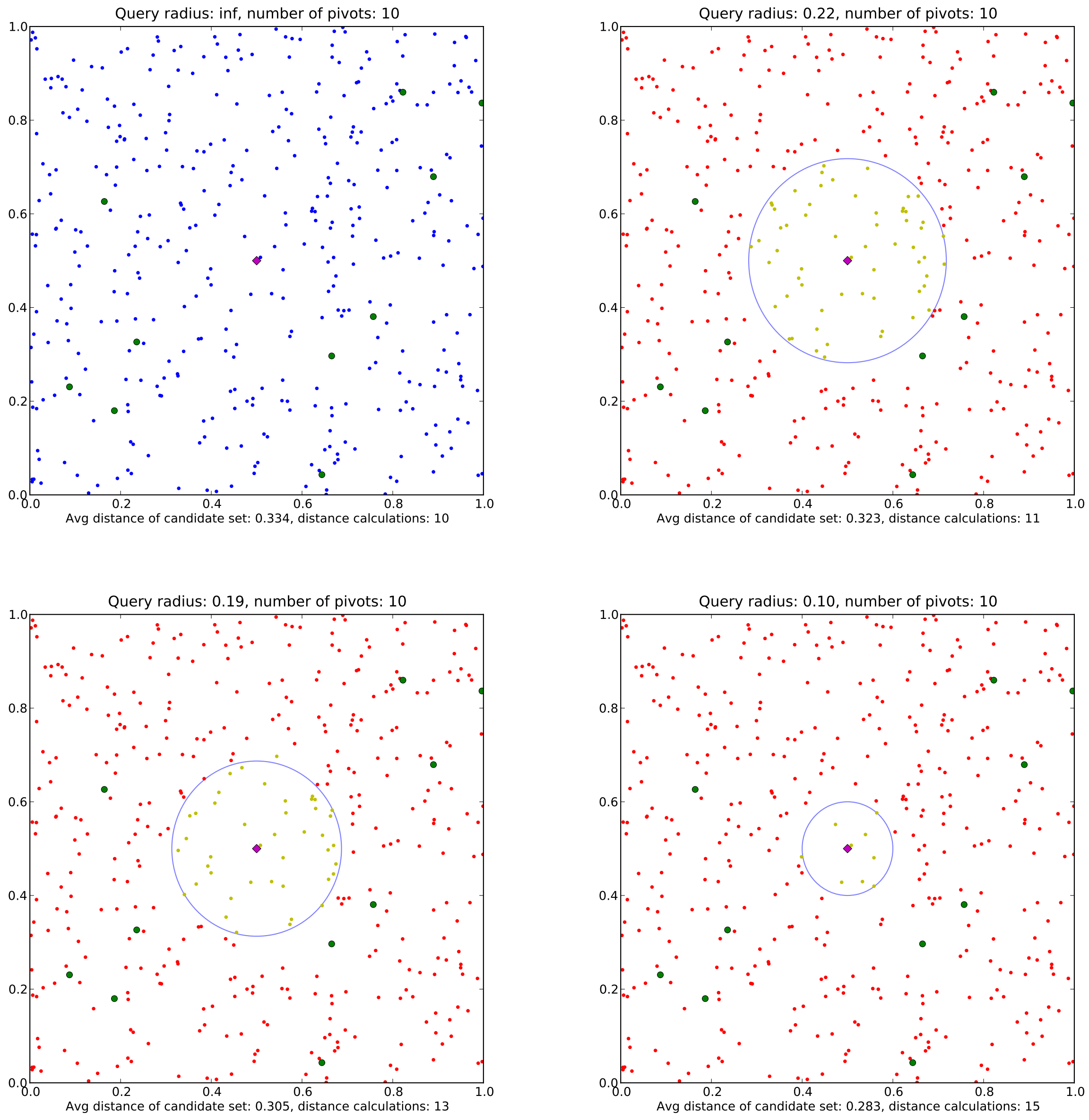


Figure 2.4: Example of kNN implemented with a LAESA range search. The radius is infinite in the first round and the k objects that are heuristically closest to the query are chosen as a candidate set. The following iterations is a 'branch and bound' search to minimize the radius until the k closest objects are found.

The appropriate size of the pivot set can be difficult to know. There are some things that can be used as a heuristic though. A higher dimensional data is more difficult to index than lower dimensional data. Using this insight one can use a measure of dimensionality, e.g. intrinsic dimensionality (see Section 2.3.2), as a heuristic for an approximation of the appropriate size of the pivot set. However, this will not give the exact size of a optimal pivot set.

Pivot selection The simplest way to select pivots is to select them at random. This has the advantage of having a very low complexity and should speed up the creation time of an index. The downside of this method is that different objects are more suitable as pivots than others and one might end up with a index that performs poorly. A more sophisticated way is to incrementally select pivots that maximize the distance between the pivots. The downside of this is the cost, after selecting the first pivot at random one have to do a linear scan for each pivot. This gives a complexity of $O(mn)$ where n is the number of pivots and m is the size of the data set that the index represents.

Another consideration is the number of pivots to use. Using to many will increase the cost of index creation and possibly search. Using to few will result in poor filtering performance of the index. Indexing "difficult" datasets requires more pivots than a "simple" one. To measure how "difficult" or "simple" a data set is one can use the data sets aintrinsic dimensionality. A data set with high intrinsic dimensionality is more difficult to successfully index than a low one. The intrinsic dimensionality can be used as a heuristic to determine how many pivots are needed. A high intrinsic dimensionality will require more pivots then a low dimensionality.

Tree based methods

SSSTree The SSSTree [9] is based on *Sparse Spatial Selection* [26], a dynamic method that selects a set of pivots that is well distributed in the metric space. The strategy it follows is to evaluate each new object in the index as a potential pivot. The object is chosen as a pivot if the distance from it to the other pivots is greater or equal to $M\alpha$, where M is the maximum distance between objects and α is a tuning constant.

The SSSTree uses this heuristic to build a tree of clusters. The first object inserted is chosen as the center of the cluster in the first node. When more objects are inserted into the tree they are either selected as the center of a new cluster or added as an object to the closest cluster. An object is chosen as a new cluster center if the distance from the object to the closest cluster center is greater than $M\alpha$. This construction process will result in a unbalanced tree. The different levels in the tree will adapt to the complexity and distribution of the objects within it [9].

2.4 Software libraries

Some software packages were used in addition to the software created specifically for this project; Yahoo! QLAS, Yahoo! Vespa, Weka and Gensim.

2.4.1 Yahoo! Query Linguistic Analysis Service

Yahoo! have developed a service *Query Linguistic Analysis Service* (QLAS) to find different possible *intents* behind search queries. Examples of this can be different things called the same, e.g. different towns around the world called "Bergen" or the fact that "New York" can be used to refer to either the state or the city.

A QLAS analysis of a query will try to find different interpretations of the query and weight each interpretation with a probability of being correct. An example of this can be found in section A.1.1, which is a QLAS analysis of the query "new york". The output of QLAS is very verbose, the following table gives a summary of the different interpretations.

Interpretation	Score
place_category: city	0.991837
place_category: state	0.982552
media_category: movie	0.346093
brand_type: manufacturer	0.161142

QLAS Output A QLAS analysis will create one or more *interpretations* of a query, six in the case of the query "new york". Each interpretation will have a *domain* field which is a notion of what the interpretation means as a whole. An example of this:

```
Interpretation:
 query=[new york]
  [domain]->[
    {
      "name":"domain_local",
      "prob":0.300649,
      "schema":"lm"
    },
    {
      "name":"domain_product",
      "prob":0.442650,
      "schema":"lm"
    }]
```

The *name*-field is the name of the domain and the *schema*-field is the kind of domain the name was taken from. QLAS has three kinds of domains, "coexistence", "language model" (lm) , and "jabba". A full overview of each domain can be found in Section A.1.2.

In addition to the domain field, an interpretation will take the given query and analyze it for different possible *spans*. If the query is in English, it is typically tokenized by white space and in addition, a span could be several words together. Continuing with the example query of "new york", this will generate three spans, "new" "york" and "new york". From the actual output from Section A.1.1:

```
Interpretation:
id=3
    span=[0,3][new] id=12 class=token referent=
    span=[4,4][york] id=13 class=token referent=
    span=[0,8][new york] id=16 class=place_name referent=


Interpretation:
id=4
    span=[0,3][new] id=12 class=token referent=
    span=[4,4][york] id=13 class=token referent=
    span=[0,8][new york] id=15 class=media_title referent=
```

As seen in this example, a span will have some `class`. This class is taken from a QLAS taxonomy which can be found in its entirety in Appendices A.1.3. The taxonomy consists of six types of classes:

- `syntactic`, e.g. *token*
- `proper_noun`, e.g. *place_name*
- `common_noun`, e.g. *business*
- `adjective`, e.g. *color*
- `code`, e.g. *company_ticker*
- `no_classification`, e.g. *compound*

Finally, the score field indicates how certain the interpretation made by QLAS is. This is a number between 0 and 1 telling how certain QLAS is that the interpretation is correct. In the New York example the interpretations "`class=place_name, [taxonomy: place_category]-> "/state"`" and "`class=brand_name, [taxonomy: brand_type]-> "/manufacturer"`" were found. The first interpretation has a score of 0.991837 and the last one has a score of 0.161142.

2.4.2 Vespa

Vespa is a search engine developed at the Yahoo! Trondheim office. Vespa can be summarized as following:

> “Vespa is a generic, scalable platform for creating search applications in Yahoo!
>
> Vespa consists of three main components:
>
> - Vespa Search - indexing and query processing
> - Vespa Document Store (VDS): document and object storage
> - Vespa Document Processing (VDP): document processing and transformation pipeline
>
> Vespa is well suited to index and search general text content as well as more structured information. Vespa supports real-time updates of full documents and massive partial updates, the latter with sub-second guarrantees. Vespa sports a highly flexible and easy-to-work with relevancy framework.
>
> Vespa is used to power most of Yahoo!'s search needs, with the exception of Web Search, currently more than 120 properties use Vespa.
>
> As of April 2009, Vespa serves close to 9 billion page views per month.
> ”

(Source: Internal Yahoo! documentation [1])

2.4.3 Gensim

Gensim [29] is an information retrieval Python library. It implements traditional information retrieval methods like *tf-idf* and *cosine similarity* (see Section 2.1) but also more advanced methods like *Latent semantic indexing*, *Latent Dirichlet Allocation* and *Random Projections*.

2.4.4 Weka

Weka is an open source machine learning software pack developed at The University of Waikato, New Zealand. This is how they describe the project [16]:

> “The overall goal of our project is to build a state-of-the-art facility for developing machine learning (ML) techniques and to apply them to

> real-world data mining problems. Our team has incorporated several standard ML techniques into a software "workbench" called WEKA, for Waikato Environment for Knowledge Analysis."

Weka enables one to rapidly experiment with many different machine learning algorithms. As stated in Section 2.2, there is no single algorithm that outperforms the rest in all learning scenarios. It is therefore valuable to be able to test whole families of algorithms to quickly identify the one that works well in the learning scenario one is pursuing.

Chapter 3

Experiments

3.1 Evaluation methodology

All the experiments in this project have the goal of correctly labeling the relevant source for a search query. The correct way to evaluate an experiment is therefore given by the training and evaluation data that the experiment utilizes. The experiments uses a data set from Yahoo! Taiwan that has a separate training and evaluation part. This data set gives a single correct source for many example queries and the evaluation is therefore the percentage of correctly classified queries from the evaluation data set. See Section 2.2.8 for further details about the training data.

The experiment in Section 3.5, using metric indexing, is focused on speeding up the classification process. To evaluate this the measurement is an achieved reduction in distance calculations (the costly part of the source classification) and how this reduction affects the classification performance.

3.2 Entropy based QLAS classification

In this experiment the objective is to classify a *query* $\rightarrow$ *source* (classify which source is relevant for a query) using compression based classification of its QLAS analysis. The basic idea is to use an entropy analysis to measure the distance from the test query QLAS object to the set of training objects. The category in the training set with the closest average distance is chosen as relevant for the query.

This experiment has many positive aspects because the value of the underlying data is unknown. It is not know if the QLAS data is a good source of information in source classification. If it has valuable information it is not known how this should be used. Entropy classification is parameter free and the training data has no form or format requirements. This means that the data can be used as is and it does not require any parameter tuning.

3.2.1 Direct zipping

This method is based on the work of Benedetto et al. [6] and can be explained as follows. The classified training data is split into their respective file set based on their class and this file set is compressed using a standard compression algorithm. This gives a file set like "$class_1.zip$", "$class_2.zip$", "$class_n.zip$".

The classification process for a new document works as follows. Add the document to each categorized file set and compress them together, giving the files "$class_1 + unknown.zip$", "$class_2 + unknown.zip$", "$class_n + unknown.zip$". The class where the difference between the compressed classified data and the compressed classified data plus the unknown document is the smallest, is chosen as the class for the new document. Equation (3.1) summarizes the process. See Section 2.2.5 for further background theory.

$$Category = \min(\Delta_{CatA:b}, \Delta_{CatB:b}, ..., \Delta_{CatX:b}) \quad (3.1)$$

$$\Delta_{CatX:b} = length(compress(CatX + b)) - length(compress(CatX)) \quad (3.2)$$

3.2.2 Normalized Compression Distance

Like the previous experiment, this method also uses a standard compression algorithm. The method was presented in Li et al. [23] and can be seen as a further development in the same field of using entropy for classification. See Section 2.2.5 for further background theory.

This experiment compares each training instance in each category independently using the function described in Equation (3.5). For each possible class, the average distance between a training instance and the unknown instance is calculated. The class with the smallest average distance is chosen as the classification of the unknown.

$$Category(unknown) = \min(CategoryA, CategoryB, CategoryX) \quad (3.3)$$

$$CategoryX(unknown) = \frac{\sum_{instance}^{CategoryX} NCD(instance, unknown)}{length(CategoryX)} \quad (3.4)$$

$$NCD(x, y) = \frac{C(xy) - \min(C(x), C(y))}{\max(C(x), C(y))} \quad (3.5)$$

3.3 Feature vector based machine learning

To evaluate the performance of methods that require feature vectors as input data, a *feature vector extractor* was created. This enables a wide array of machine learning algorithms. This extractor takes the full semistructured QLAS data (see Appendix A.1.1) that was used in the previous experiment and creates a feature vector in the form of Table 3.1. The feature extractor creates a single feature vector for each interpretation in a QLAS analysis. Table 3.1 shows how two instances from the training set with their QLAS data are transformed to seven feature vectors.

It should be noted that this is a naive way to do a feature vector extraction. The task of creating good automatic feature vector extractors is a field of its own. The result of this experiment can therefore be seen as an evaluation of a naive feature vector extraction, as much as the actual algorithms.

The algorithms in the following list was tested, using their implementation in Weka (see Section 2.4.4).

- Naive Bayes (see Section 2.2.6)
- Decision tables + Naive Bayes (Section 2.2.6)
- IB1 (see Section 2.2.6)
- RIPPER (see Section 2.2.6, named JRip in Weka)
- C4.5 (see Section 2.2.6)
- Random Forest (see Section 2.2.6)

The goals of this experiment are the same as those in Section 3.2, but with alternative methods. The result of this experiment is given in Section 5.2 and compared to the experiments in Section 3.2 in Figure 5.3.

3.4 Traditional information retrieval similarity

This section presents two experiments using two different search engines. The idea behind using search engines in this context is to use the ranking feature in the search engine to find the most relevant QLAS analysis. Those QLAS documents are then used to classify a query.

original

class	query	lang	prob_a	prob_b	prob_c		QLAS
blog	樹林地名由來	zh-hant	-8.025	-26.071	-13.283	0	QLAS OBJECT

extended

class	query	lang	prob. a	prob. b	prob. c	qlas class	class prob	qlas taxonomy	taxon. score	taxon. desc
blog	樹林地名由來	zh-hant	-8.025	-26.071	-13.283	domain_ product_ tw	0.004	place_ name	0.518	place_ category_ city
blog	樹林地名由來	zh-hant	-8.025	-26.071	-13.283	domain_ product_ tw	0.004	?	?	?
blog	樹林地名由來	zh-hant	-8.025	-26.071	-13.283	domain_ product_ tw	0.004	media_ title	0.062	media_ category_ song

original

class	query	lang	prob_a	prob_b	prob_c		QLAS
blog	壽山高中	ja	-7.243	-19.553	-11.040	0	QLAS OBJECT

extended

class	query	lang	prob. a	prob. b	prob. c	qlas class	class prob	qlas taxonomy	taxon. score	taxon. desc
blog	壽山高中	ja	-7.243	-19.553	-11.040	domain_ product_ tw	0.033	business	0.785	industry_ educational
blog	壽山高中	ja	-7.243	-19.553	-11.040	domain_ product_ tw	0.033	organization_ name	0.284	industry_ sports_ recreation/attraction
blog	壽山高中	ja	-7.243	-19.553	-11.040	domain_ product_ tw	0.033	organization_ name	0.185	industry_ sports_ recreation/attraction
blog	壽山高中	ja	-7.243	-19.553	-11.040	domain_ product_ tw	0.033	?	?	?

Table 3.1: Augmented feature vectors. Two single vector in the tables labeled *original* has been expanded to three and four vectors, each containing one QLAS interpretation.

3.4.1 Off-the-shelf search engine

This experiment uses a standard (proprietary) search engine to measure the similarity between QLAS documents. This is done using the search engine Vespa, described in Section 2.4.2.

The experiment is performed in the following way: all the QLAS documents from the training set is indexed as a normal text document together with their classification. To classify a new query a two step process is performed. It is first analyzed by QLAS, then the QLAS document is used to construct a query. The ten best results are retrieved by Vespa and the average relevance for each category is calculated. An example output of the search result can be found in Section A.2.7. The class with the highest average relevance is selected as the class for the new query.

If this experiment should work it has some big advantages. It uses off-the-shelf components that are already optimized for performance. However, a search engine similarity function is made with human queries and natural language documents in mind. It is therefore reasonable to be pessimistic about the performance.

3.4.2 Traditional information retrieval: cosine distance, tf-idf

The experiment in the previous section has a major shortcoming. A proprietary search engine can be seen as a "black box" because the internal mechanisms are not readily available. If the experiment is a success or failure it is hard to reason and analyze the underlying cause of the result. Because of this, a second experiment was done with a "textbook" implementation of the vector model, tf-idf and cosine distance. This was done using the Python information retrieval library *Gensim* [29].

The experiment works much in the same way as the one described in the previous section. Each QLAS document is transformed into a vector and put in a matrix consisting of all the document vectors of the document class. A search is then performed in each of the document class indexes and the result from each is joined into a single result. The 10 closest hits was then used to classify the search query using the functions in Section 3.5.2.

3.5 Metric indexing

The entropy based experiment in Section 3.2 is computationally very expensive. The method calculates the expensive distance function from the query object to all the objects in the training set. The category with the lowest average distance from the query object is chosen as the classified category. This is in other words, a *linear scan* of the training data.

The goal of this experiment is to evaluate if metric indexing can be applied to the problem of classifying sources using QLAS data. To evaluate this, a significant speedup measured in fewer distance calculations should be observed. This reduction in cost should not affect the positive result in experiment 5.1.2 significantly.

3.5.1 Distance measurement

The distance measurement in metric indexing is usually a proper metric. This is however not the case in this experiment. Even though it is not a proper metric it is still a distance measurement and the though is to use it as an "approximated metric" to get a approximated search. This can however not guarantee that the closest objects are returned in a kNN search.

NCD

There is a problem when using NCD as a metric in metric indexing, *it is not a proper metric.* NCD is only a proper metric when using the actual Kolmogorov complexity, which is not computable. The complexity is approximated using a practical available compression algorithm. Hence, NCD is in practical terms an approximation of a proper metric. This does not mean that it will not work at all under any circumstances, but it means that using metric indexing to find the k nearest neighbors the index can not guarantee that the returned objects are the k closest.

3.5.2 kNN

When using kNN to do classification one has do decide how to interpret the returned closest objects. This is a general statement about kNN, be it in regards to metric indexing, case based reasoning or any other application. There are different ways to do this. One can calculate the distance to the k elements and choose the class that has the lowest average distance or one can use each objects as a "vote", selecting the class that has the most objects in k. The selected classification function should be selected in the basis in what the data set represents.

In this case we are interested in the normalized compression distance of the k nearest objects where closer means better and more objects of the same class should speak to the correctness of the class. Three functions were tested to classify the result, listed in the following table.

Function	**Explanation**
Voting	The class with the majority of objects among the k closest are chosen. This function does not make sense if $k = n$.
Average distance	The class with the lowest average distance to the k objects is chosen. This function was also used in the entropy experiment in Section 3.2.
Weighted average distance	The class with the lowest average distance multiplied by how many percent of k the class populates. This function only makes sense to uses if $k \neq n$. The reasoning behind it is that a single object should not be chosen if there are several other objects in another class that are also reasonable close.

3.5.3 Index structures

As stated in Section 2.3, metric indexing have several possible indexing methods and similarity metrics. Two index structures were used in this experiment, LAESA and SSSTree. The LAESA experiment used 32 and 96 random pivots. See Section 2.3.3 and 2.3.3 for the background theory about these.

Chapter 4

Implementation

Some software were written to perform the experiments in this project. Some 3rd-party packages were also used. This chapter describes the software written from scratch. All the software was written in Python, but Python modules written in C were extensively used.

4.1 Entropy classification

The entropy classification extensively uses the Python modules *zlib* and *bz2*. The classification process is easily done in parallel (no synchronization needed between threads) so the program was written in parallel using the Python classes *threading.Thread* and *Queue.Queue*. The classification process is started with loading all the queries into a queue and the training corpus into a dictionary. From there the following is done in each thread:

1. Retrieve the QLAS analysis of the query.
2. Calculate the entropy using either Equation 4.1 or Equation 4.2 ($C(x)$ is a compression function).
3. Select the class with the lowest entropy as the classification for the unknown instance.
4. Save the classification in the a result queue.

In addition to the classification threads a thread is created to collect the result from the classification threads. When n new classification have been collected the result is saved to permanent storage. The number of classification threads was set equal to the number of CPU cores.

$$\begin{aligned} NCDCategory(unknown) &= \frac{\sum_{instance}^{Category} NCD(instance, unknown)}{length(Category)} \\ NCD(x,y) &= \frac{C(xy) - \min(C(x),C(y))}{\max(C(x),C(y))} \end{aligned} \tag{4.1}$$

$$\begin{aligned} DirectZip(unknown) &= length(compress(Category + unknown)) - \\ &\quad length(compress(Category)) \end{aligned} \tag{4.2}$$

4.2 Feature vector classification

The feature vector classification was done in Weka (see Section 2.4.4) so those algorithm were not implemented from scratch. However, to be able to use Weka and those algorithm the QLAS data have to be transformed from the raw QLAS data to feature vectors. A program was written to do this. The task is a straight forward parsing task. If one looks at the example QLAS analysis in Appendix A.1.1, it has a very clear form. The first part is some meta data and then a list of interpretations. Inside the interpretation there is also a repeating structure. To parse the document the program first split the document in individual interpretation. After that each interpretation is parsed.

The program defines a QLAS class that represents the information and contains the logic to parse the raw data. After the parsing is done the feature vectors representing the QLAS object can be extracted as Python lists. These lists was stored to disk as comma separated values, using the module *csv*. An example of how the program takes one QLAS analysis and create several feature vectors can be found in Table 3.1.

4.3 Traditional information retrieval classification

This experiment was also performed using two 3rd-party software packages, Vespa and Gensim. Vespa is a proprietary search engine developed at Yahoo! Technologies Norway. Gensim is a open source information retrieval package written in Python. The classification process is easily parallelized in the same way as the program described in Section 4.1. The following was done in each classification thread:

1. Retrieve the QLAS analysis of the unclassified query.
2. Use the QLAS data to create a query. This was done by removing all non-textual characters (parentheses, colons and so one) and then take every word in the remaining text to create a query like $term_1||term_2||..||term_n$ where $||$, means OR.
3. Submit the query and use the search result to classify the query.

4.4 Metric indexing

Two metric indexing techniques was implemented, LAESA and SSSTree.

4.4.1 LAESA

The indexing and search in LAESA was implemented using a Numpy [21] matrix and Numpy matrix operations.

Index creation

A LAESA index is a $n \times m$ matrix where n is the number of objects in the collection that is indexed and m is the number of pivots. To be able to do matrix operations efficiently the matrix was stored in a Numpy matrix.

To create an index m random pivots was chosen, $m/4$ from each class. The NCD (using *zlib*, compression level 9) distance from each object to each pivot was calculated and stored in the matrix. The index matrix was then saved together with the index id of the pivots. The Python module *pickle* was used to serialize and save the data.

Searching

To do a search the entire index is loaded into memory. Two search types were implemented, range search and kNN search.

A range search with radius r is initiated by calculating the distance to the pivot elements. This, together with the index gives the upper bound $\hat{d}(q, x)$ and the lower bound $\check{d}(q, x)$. This is done quickly by adding/subtracting the distance vector *query* $\rightarrow$ *pivot* to the whole index. The objects are separated into three groups, *included*, *discarded* and *maybe*. The distance from the query object to the objects in the *maybe* group is then calculated. The *included* group and the objects in the *maybe* group that have a distance lower then r are returned.

The kNN search is implemented using the range search, but the distances to the maybe group is not calculated immediately. A range search with $radius = \infty$ is started and the distance to k object is calculated. The largest distance of the k object is then used as the new radius. A new range search is then done. If a new object is found within the new radius, it is added to the set of k possible objects and the object in the set of k with the greatest distance is removed from the set. This process of decreasing the radius is continued until the search can not be improved anymore.

4.4.2 SSSTree

The SSSTree index is a unbalanced tree of clusters. This was implemented using Python objects where each object contains a node center and a list of child nodes.

Index creation

The index creation is started by creating a single node using the first object in the object collection. When the next objects are added a recursive process is started. It is first added to the root node. At this level it is decided if the object should be added to a new child node under the root, or adding it to one of the child nodes already created. This process is continued recursively if it is added to an existing child node.

There are two scenarios when adding an object to a node; either adding it to the root node or adding it further down the tree. The difference in adding a node to the root node or a child node is the calculation used to estimate the space covered by a node. The function defined in Equation 4.3 is used in the root node. The function defined in Equation 4.4 is used in child nodes. If the distance is greater then respectively Equation 4.3 or Equation 4.4 is it added as a new child node. If the distance is smaller it is added to the closest existing child node.

$$\begin{aligned} distance(root\_center, new\_object) &\geq M \times \alpha && (4.3)\\ distance(node\_center, new\_object) &\geq covering\_radius \times \alpha && (4.4) \end{aligned}$$

$$covering\_radius = 2 \times \max_{child \in ChildNodes} distance(node\_center, child) \qquad (4.5)$$

The M variable used at the root is the greatest possible distance between objects. The $covering\_radius$ variable (defined in Equation 4.5) is an estimation of the space covered by a node. α is tuning constant, usually set around 0.35 to 0.4. When creating a *NCD* SSSTree index it turned out that a slightly higher α was needed, 0.48 worked well.

Searching

The search process in a SSSTree has the goal of pruning parts of the tree. This is done by using Equation 4.6 at nodes recursively. A node and its children can be pruned from the search if the distance from the query object to the node center, minus the radius, is greater than the covering radius multiplied with the alpha constant.

The search was implemented by having a search method at each node object. The search method is called recursively on each node that could not be pruned from the search. The returning result from each child node is concatenated and returned higher up the tree.

$$distance(query, node\_center) - radius \quad \geq \quad covering\_radius \times \alpha \tag{4.6}$$

Chapter 5

Results

The experiments in this project used 138 699 training cases and 15 164 test cases evenly divided into four classes; *blog*, *image*, *knowledge* and *shopping*. Completely random classification should be 25% correct so any result close to that would not be a positive result. See Section 2.2.8 for further details about the training data.

Each result table is a 5×7 matrix. The left column shows the class of the test set and column 3 to 6 show how the test instances were classified. An example: the first line of the first table in Section 5.1.1 show how the *blog* category was classified using direct compression / zlib; 1662 correctly as *blog*, 842 as *image*, 684 as *shopping* and 540 as *knowledge*. The rightmost column show the percent of test instances that was correctly classified.

5.1 Entropy / compression classification

This section presents the classification results using the entropy based methods *Direct Zipping* and *Normalized Compression Distance*. Please see Section 3.2 for further details about the experiment. Three levels of compression were used for comparison sake.

5.1.1 Direct compression

This experiment uses the compression libraries *zlib*[13] and *bzip2*[34]. *zlib* implements the compression algorithm *DEFLATE*, which uses a combination of *LZ77* and *Huffman coding*. *bzip2* is a block compression algorithm. It is considered a more powerful compression algorithm (higher level of compression) than zlib but is considerably slower.

The three following subsections show the results using three levels of compression. They are all better than random classification, however there are some unexpected results. One would think that a higher level of compression always gave a better result, at a computational cost. This is not true for all the experiments. The test set for the category `blog` gives the best result at level 5 (then level 1, then level 9 (the most expensive) at third place). Table 5.1.1 summarizes the correctness at the different levels. Another unexpected result is the very different behavior of *zlib* and *bzip2*. This is discussed further in Section 6.1.1.

Level 1 zlib/bzip results

Result using level 1 zlib/bzip2 compression as a means of classification.

zlib:

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**1662**	842	684	540	44.58%
`image`	3862	771	**2157**	502	432	55.85%
`shopping`	3749	662	849	**1996**	314	52.24%
`knowledge`	3821	756	693	595	**1705**	44.62%

bzip2:

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**32**	0	2111	1585	0.86%
`image`	3862	32	**0**	2189	1641	0.00%
`shopping`	3749	17	0	**2366**	1438	61.92%
`knowledge`	3821	32	1	2043	**1673**	44.63%

Level 5 zlib/bzip results

Result using level 5 zlib/bzip2 compression as a means of classification.

zlib:

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**1648**	525	1082	473	44.21%
`image`	3862	654	**1515**	1102	591	39.23%
`shopping`	3749	461	499	**2606**	255	68.20%
`knowledge`	3821	657	468	754	**1870**	49.88%

bzip2:

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**6**	934	2125	663	0.16%
`image`	3862	2	**1038**	2239	583	26.88%
`shopping`	3749	1	972	**2354**	494	61.61%
`knowledge`	3821	5	1000	2146	**598**	15.95%

Level 9 zlib/bzip results

Result using level 9 zlib/bzip2 compression as a means of classification.

zlib:

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**1020**	913	1023	772	27.36%
`image`	3862	253	**2013**	962	634	52.12%
`shopping`	3749	198	579	**2741**	303	71.74%
`knowledge`	3821	236	505	649	**2359**	62.92%

bzip2:

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**3003**	162	95	468	80.55%
`image`	3862	3155	**150**	127	430	3.88%
`shopping`	3749	3067	165	**151**	438	3.95%
`knowledge`	3821	3080	154	109	**406**	10.83%

Levels compared

Classification results using different levels.

zlib:

Category	Level 1	Level 5	Level 9
`blog`	44.58%	44.21%	27.36%
`image`	55.85%	39.23%	52.12%
`shopping`	52.24%	68.20%	71.74%
`knowledge`	45.48%	49.88%	62.92%

bzip2:

Category	Level 1	Level 5	Level 9
`blog`	0.86%	0.16%	80.55%
`image`	0.00%	26.88%	3.88%
`shopping`	61.92%	61.61%	3.95%
`knowledge`	44.63%	15.95%	10.83%

5.1.2 Normalized Compression Distance

This experiment is similar to the previous, but it uses the *Normalized Compression Distance* to measure entropy. A difference in this experiment, in addition to using a different distance measurement is that the experiment evaluates each case in the training corpus individually. The class with the lowest average distance is chosen as the class for the unknown instance.See Section 3.2.2 for a full description of the experiment.

The following sections summarize the result using three levels of compression. The results are in general much better than those of Section 5.1.1 but this method is considerably slower. The reason for this speed difference is that the method compresses individual training cases and the unknown test one by one. The previous method compresses the whole training corpus for each class in one swoop.

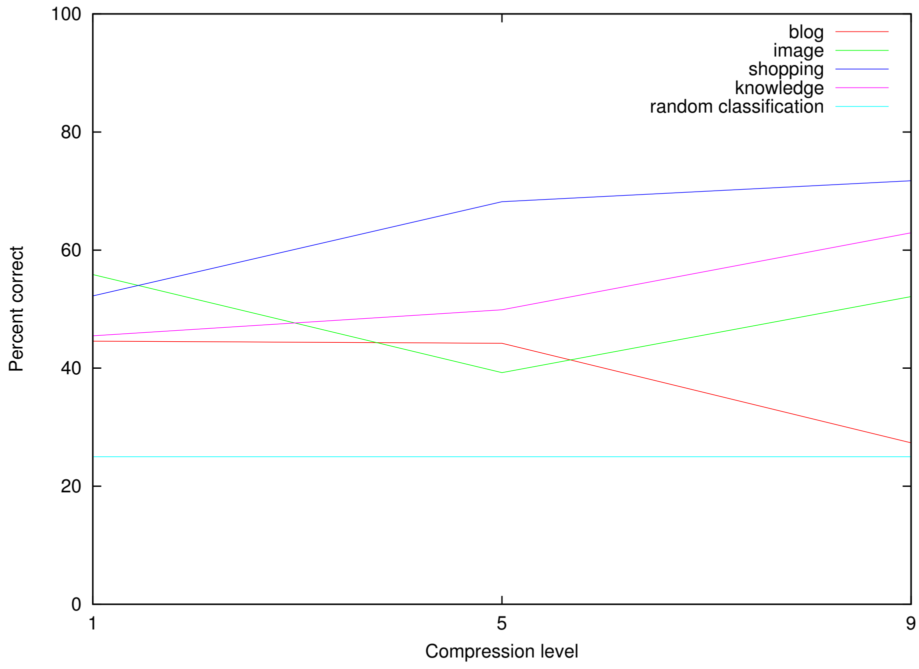
zlib, compression level 1

Result using NCD with level 1 zlib.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**2064**	358	1174	132	55.36%
`image`	3862	0	**3014**	424	424	78.04%
`shopping`	3749	0	0	**3777**	44	98.85%
`knowledge`	3821	0	732	321	**2696**	71.91%

zlib, compression level 5

Result using NCD with level 5 zlib.



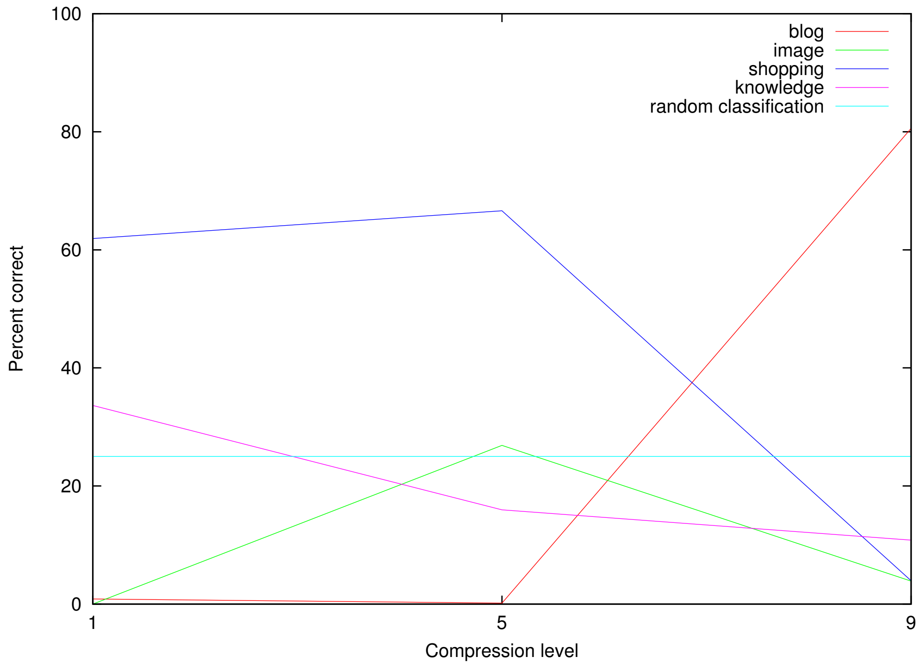


Figure 5.1: Plot of three compression levels using `zlib` and `bzip2` classification.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**1870**	345	1173	340	50.16%
`image`	3862	0	**2940**	377	545	76.13%
`shopping`	3749	0	0	**3754**	67	98.25%
`knowledge`	3821	0	708	311	**2730**	72.82%

zlib, compression level 9

Result using *NCD* with level 9 zlib.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	3728	**1786**	353	1217	372	47.91%
`image`	3862	0	**2939**	381	542	76.10%
`shopping`	3749	0	0	**3757**	64	98.33%
`knowledge`	3821	0	718	321	**2710**	72.29%

Levels compared

Category	Level 1	Level 5	Level 9
`blog`	55.36%	50.16%	47.91%
`image`	78.04%	76.13%	76.10%
`shopping`	98.85%	98.25%	98.33%
`knowledge`	71.91%	72.82%	72.29%

Table 5.1: Classification results using NCD at different levels.

5.2 Feature vector based machine learning

These sections presents the classification result using the feature vector based learning algorithms presented in Section 2.2.6. The results can be evaluated from two perspectives; the performance of the algorithms and the quality of the feature vectors. This is discussed further in Section 6.2. A graphical comparison of the feature vector algorithms and *Normalized Compression distance* can be found in Figure 5.3.

One thing to notice is the different number of test (and training) instances in the results. The numbers are different because each instance in the original corpus

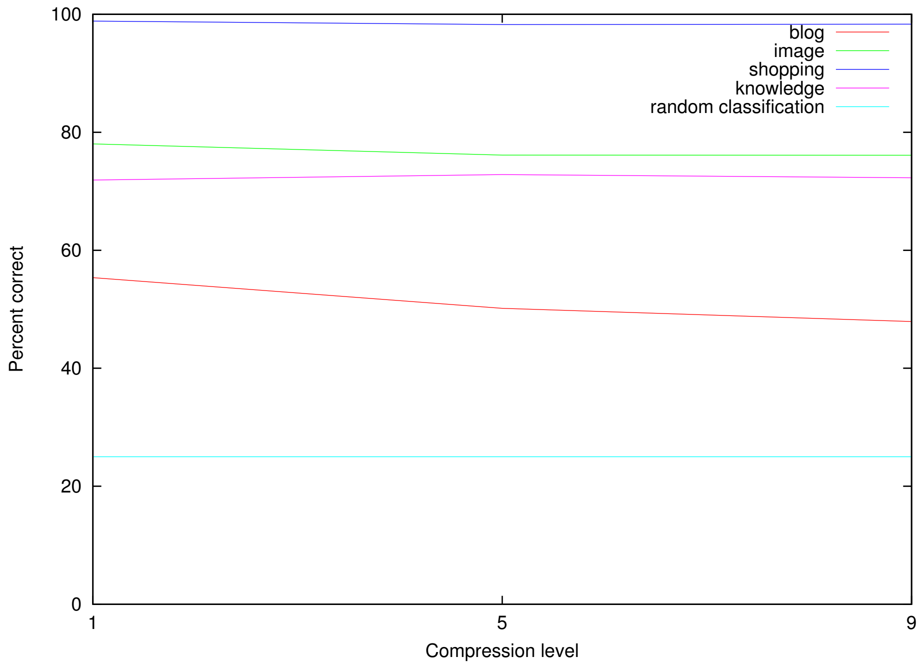


Figure 5.2: Plot of three compression levels using NCD/`zlib` classification.

is a pair (query, QLAS data for query), and a few more numbers associated with the query. Each QLAS analysis contains several interpretations of the query they represent. Each of those interpretations are used to create a individual feature vector. More about this process can be found in Section 3.3.

5.2.1 Naive Bayes

Full result output in Appendix A.2.1.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	12987	**1018**	2248	5361	4360	7.8%
`image`	13365	548	**3481**	4609	4727	26%
`shopping`	13848	758	2740	**7003**	3347	50.6%
`knowledge`	12906	417	1916	2342	**8231**	63.8%

5.2.2 DTNB

Full result output in Appendix A.2.2.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	12987	**3690**	2444	3208	3645	28.4%
`image`	13365	2033	**4236**	2992	4104	31.7%
`shopping`	13848	2748	2847	**5163**	3090	37.3%
`knowledge`	12906	1238	2024	1435	**8209**	63.6%

5.2.3 IB1

Full result output in Appendix A.2.3.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	12987	**6499**	2430	2672	1386	5%
`image`	13365	4830	**4037**	2577	1921	30.2%
`shopping`	13848	5361	3024	**4071**	1392	29.4%
`knowledge`	12906	4622	3056	1786	**3442**	26.7%

5.2.4 RIPPER

Full result output in Appendix A.2.4.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	12987	**358**	67	11758	804	2.8%
`image`	13365	54	**279**	11681	1351	2.1%
`shopping`	13848	289	80	**12630**	849	91.2%
`knowledge`	12906	23	46	9591	**3246**	25.2%

5.2.5 C4.5

Full result output in Appendix A.2.5.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	12987	**4304**	2320	3049	3314	33.1%
`image`	13365	2522	**3702**	2851	4290	27.7%
`shopping`	13848	3186	2760	**5019**	2883	36.2%
`knowledge`	12906	1683	1743	1222	**8258**	64%

5.2.6 Random Forest

Full result output in Appendix A.2.6.

		Classified category				
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	Correct
`blog`	12987	**4274**	2161	3597	2955	32.9%
`image`	13365	2126	**4280**	2995	3964	32%
`shopping`	13848	2870	2427	**5853**	2698	42.3%
`knowledge`	12906	1447	1882	1232	**8345**	64.7%

5.3 Traditional information retrieval similarity

This section present the results of the information retrieval experiments. The idea here is to use a regular text search engine to find similar QLAS documents. See Section 3.4 for further details.

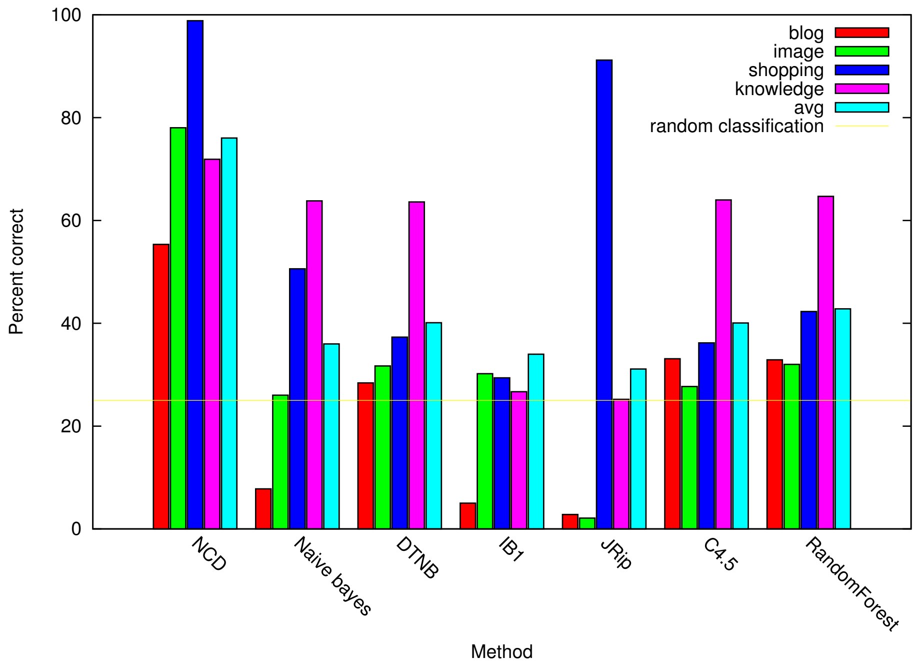


Figure 5.3: A performance chart of all the feature vector based methods and *Normalized Compression Distance*

5.3.1 Vespa classification

The result of the experiment presented in Section 3.4.1. Vespa was set up to return the 10 closest objects.

Vespa had the option to not return any results. This leads to a dilemma of how to interpret the correctness of the classification. One option is to view a empty result as the wrong classification, the test instance is wrongly classified as the class *no-class*. Another option is to ignore test instances that were not classified from the total set of instances (the sum of tests does not include the unclassified). It is not clear what is correct so both numbers are included in the *Correct* column. The first number is the correctness if unclassified is included, the second is without.

		Classified category					
Test category	Tests	`blog`	`image`	`shopping`	`knowledge`	`no class`	Correct
`blog`	3728	**1105**	396	400	1542	285	29.6% / 32.1%
`shopping`	3862	1142	**495**	422	1620	183	12.8% / 13.5%
`image`	3821	1112	464	**399**	1702	144	10.4% / 10.9%
`knowledge`	3749	1222	456	396	**1323**	352	35.3% / 38.9%

5.3.2 Gensim classification

The result of the experiment presented in Section 3.4.2. This experiment is a kNN with $k = 10$. The k closest objects was classified using three classification methods. These are presented in Section 3.5.2.

Voting classification

		Classified category				
Test category	Tests	blog	image	shopping	knowledge	correct
blog	3725	**2966**	340	248	171	79.6%
image	3862	2568	**848**	266	180	22.0%
shopping	3821	2482	431	**774**	134	20.3%
knowledge	3749	2539	428	273	**509**	13.6%

Average distance classification

		Classified category				
Test category	Tests	blog	image	shopping	knowledge	correct
blog	3725	**2978**	236	244	267	79.9%
image	3862	2506	**837**	274	245	21.7%
shopping	3821	2494	239	**886**	202	23.2%
knowledge	3749	2442	269	236	**802**	21.4%

Weighted average distance classification

		Classified category				
Test category	Tests	blog	image	shopping	knowledge	correct
blog	3725	**2893**	337	273	222	77.7%
image	3862	2427	**911**	294	230	23.6%
shopping	3821	2356	395	**892**	178	23.3%
knowledge	3749	2408	398	280	**663**	17.7%

5.4 Metric indexing

This section present the result of using metric indexing to speed up the classification process using *NCD* (using *zlib*, compression level 9) as a metric. Each result consists of two parts, the classification performance and the reduction in distance calculations. The kNN search in the metric index was set up with $k = 10$. The choice of k is discussed in Section 6.3. See Section 3.5 for further information about the experiment.

5.4.1 LAESA using 32 random pivots

This subsection presents the classification and filtering performance using *NCD* and a LAESA index using 32 random pivots.

Voting Classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**2003**	554	718	453	53.7%
shopping	3862	1069	**1480**	1141	172	38.7%
image	3749	1155	789	**1450**	355	41.5%
knowledge	3821	1762	521	864	**674**	17.4%

Average Distance Classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**1517**	569	776	866	40.7%
shopping	3862	840	**1531**	1030	461	40.1%
image	3749	901	769	**1413**	666	40.3%
knowledge	3821	1101	564	851	**1305**	34.3%

Weighted Average Distance Classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**1681**	624	912	511	45.1%
shopping	3862	1222	**1239**	1122	279	32.4%
image	3749	1166	807	**1325**	451	37.5%
knowledge	3821	1572	599	991	**659**	17.1%

On average, **62442.2** fewer distance calculations were executed at each search, giving a reduction of **45.02**% compared to a linear scan.

5.4.2 LAESA using 96 random pivots

This subsection presents the classification and filtering performance using *NCD* and a LAESA index using 96 random pivots.

Voting classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**1926**	569	742	491	54.7%
shopping	3862	1148	**1397**	1068	249	37.6%
image	3749	1289	716	**1557**	187	40.5%
knowledge	3821	1795	504	874	**678**	17.2%

Average distance classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**1442**	618	775	893	40.2%
shopping	3862	831	**1479**	1021	530	40.0%
image	3749	875	740	**1482**	652	39.0%
knowledge	3821	1104	572	865	**1310**	34.5%

Weighted average distance classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**1625**	619	927	557	45.7%
shopping	3862	1262	**1160**	1083	358	30.7%
image	3749	1304	748	**1376**	321	36.3%
knowledge	3821	1605	565	1013	**668**	17.0%

On average, **65646.3** fewer distance calculations were excluded at each search, giving a reduction of **47.33**% compared to a linear scan.

5.4.3 SSSTre

This subsection presents the classification and filtering performance using *NCD* and a SSSTree index.

Voting classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**1343**	866	856	663	35.8%
shopping	3862	625	**1973**	998	266	51.3%
image	3749	680	1028	**1576**	465	41.2%
knowledge	3821	1100	760	988	**973**	25.5%

Average Distance Classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**1475**	571	757	925	39.5%
shopping	3862	753	**1566**	965	578	40.7%
image	3749	836	730	**1464**	719	38.6%
knowledge	3821	1105	606	777	**1333**	35.1%

Weighted Average Distance Classification

		Classified category				
Test category	Tests	`blog`	`shopping`	`image`	`knowledge`	correct
blog	3728	**1082**	905	920	821	29.0%
shopping	3862	663	**1755**	1004	440	45.7%
image	3749	645	1087	**1455**	562	38.4%
knowledge	3821	858	852	1003	**1108**	29.1%

On average, **31558.8** fewer distance calculations were executed at each search, giving a reduction of **22.75**% compared to a linear scan.

Chapter 6

Result analysis and comparison

6.1 Compression based classification

The compression based methods was able to get significantly better results than what one would expect of random classification. A random classification is expected to give 25% correct. The compression based methods achieved from 55% to 98.5% correct classification in the different document classes.

There is large variation in the performance of the different categories. The reason for this, considering how the method uses entropy, can probably be attributed to a varying quality in the underlying training data. The QLAS analysis enumerates all the different possible interpretations for a query together with a probability. There is therefore no guarantee that the different categories in a training set which is augmented with QLAS is of equal quality, even if the training sets were equal before the addition of QLAS data. This will depend on the queries in each category and what QLAS can say about these queries.

It is quite reasonable to think that this method was not able to pick up on the connected interpretation and probability. A simple scenario of this would be two documents: “long_string: 0.9”, “long_string 0.1”. The normalized compression distance (Section 3.2.2) would be small because they are, from a compression algorithms perspective, very similar. However, they might not be related to the same category of documents as they have a very different relation to “long_string”. Another issue is when there are many interpretations in a QLAS document. Given that the analysis has found one interpretation that is highly probable and others that it not probable. This method might not able to deduce that the interpretation with a high probability are more important than the ones with a low probability.

The less important interpretations will then create a form of noise by taking the focus away from the important one.

6.1.1 Direct zipping

Using bzip2 and the *direct zipping* method (Section 5.1.1) gave some strange results (see Section 5.1.1). The expected result was to be monotonically increasing with a higher compression level. This was not the case. To check for possible errors an experimental run was done to verify that a higher compression level actually gave better compression on the experiment data. It did so. With some further investigation it became clear that bzip2 had some unexpected characteristics. In certain cases the size of the compressed training data were larger than the size of the compressed training data plus the unknown testing sample.

$$size(compress(training)) > size(compress(training + unknown))$$

Should this result be correct, either from the perspective of a practical compression algorithm or the theoretical Kolmogorov complexity of the data? This problem could be stated more clearly, can the Kolmogorov complexity of the prefix of a string be greater than the string itself? The intuition is that this should be possible. Let S be a long random string of high complexity and D be S repeated n times. Now let *PrefixD* be the same as D, only a few characters shorter. In this case the complexity of D is "repeat S n times" and the complexity of *PrefixD* would be "repeat S n times, then subtract a few characters". In this case *len(D)* $>$ *len(PrefixD)* and *complexity(D)* $<$ *complexity(PrefixD)*. See Figure 6.1 for a graphical representation of this. The fact that the complexity of a prefix is bigger than the full string does not affect the reasoning behind the experiment. If one is able to add a string to as long string and get a smaller complexity, it is from a entropy classification perspective, highly related.

Looking at the *bzip2* result tables in Section 5.1.1 it is clear that there is a bias for certain categories in each compression level. This bias can be explained as follows. Compressing each training set will result in a set $compress(a + testinstance) < compress(b + testinstance) < compress(c + testinstance)$. If this order changes at different compression levels it implies that the compression algorithm is able to exploit different structures in the files at different compression levels. This order change is to be expected. However, if the order change is "consistent", meaning a single category is usually preferred at a certain compression level, then you have a strong bias that explains the result of this experiment. A small test was done to test this bias. A few test instances was compressed/classified in the say way as this experiment. After each test the growth[1] for each class was examined. The result of this was the same classification pattern at different compression levels, as those in Section 5.1.1.

[1]The size of $compress(unknown + category_x) - compress(category_x)$

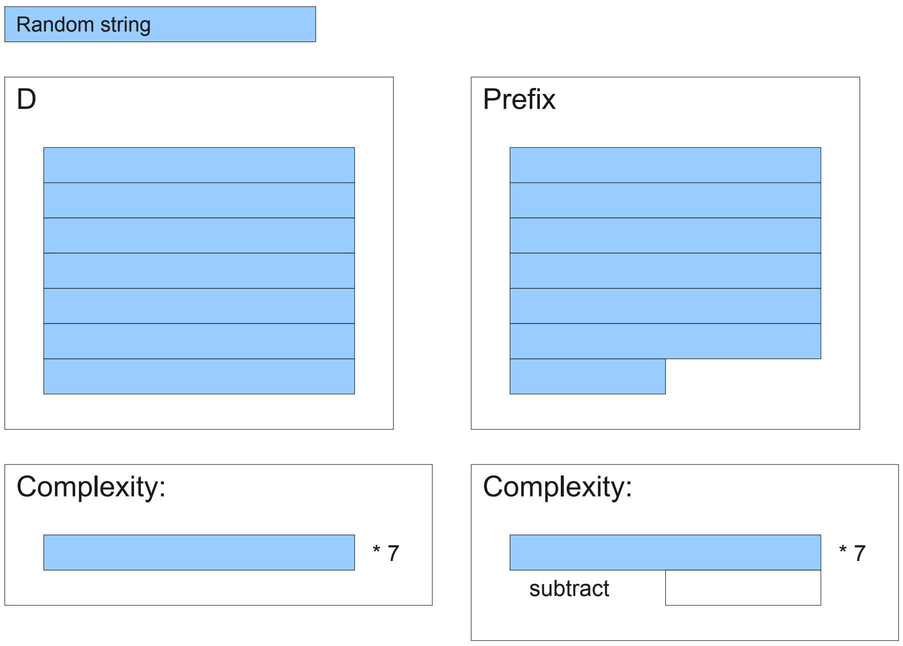


Figure 6.1: A hypothetical string and its prefix where the prefix has a larger complexity than the string it self.

From the result in Section 5.1.1, it seems that *bipz2* favors certain categories at different compression levels. One can imagine several reasons for this, using direct zipping. This method uses the size difference of $compress(category_n)$ and $compress(category_n + unknown)$ to classify an unknown case. If one does not ensure that each $category_n$ is of equal information quality (from the compressor's perspective) there could be an uneven bias for some categories. The experiment in Section 3.2.2 should be less prone to this because of its normalization. The result of the experiment supports this.

A simple example where direct zipping would fall short is if the size of each class differs much. Say that one has two files representing two classes, $class_a$ and $class_b$, and $class_b$ is much larger than $class_a$. The big file, $class_b$ could then "absorb" the test instance much better. The reasoning behind this is that a big file have a greater chance of having data pieces in common with a new small file, than the small file. This will result in a smaller increase in file size after it is compressed because compression algorithms take advantage of data similarities like this.

The scenario of varying file sized is valid for this project if QLAS generally has more information about the queries in one class than another. It does however not explain the differing bias at different compression levels.

6.1.2 Normalized compression distance

The normalized compression distance clearly gave the best classification performance with an average correctness of 76.1% (see Section 5.1.2). A surprising part of the result was nevertheless that the best result was not achieved with the highest level of compression. This is not the result one would expect. The better a compression algorithm compresses, the closer the result is to the actual Kolmogorov complexity.

The reason for the non-monotonic increase in classification performance can probably be attributed to the QLAS data that was used for the classification. As previously discussed, there is no guarantee that the QLAS analysis for different queries and sets of queries in each category is of equal information quality. Both this and the previous experiment indicate that the compression algorithm had a different bias to different categories at each compression level. This bias did, however, impact the result only slightly.

6.2 Feature vector based classification

A big part of the classification performance in the feature vector based algorithms will be given by the quality of the information stored in the feature vectors. Because there is no clear one-to-one relationship between the training data in this experiment and the entropy based experiment, it would not be right to unequivocally claim the superiority of one method over the other. The experiment can be evaluated from two perspectives; the algorithms and the quality of the feature vector extractor.

When using feature vectors the learning algorithms has to use its inductive bias to make a generalization from the training cases to a learned model. When doing this the learning algorithm is sensitive to *noise* and *overfitting* (see Section 2.2.4). Because the training data was used indiscriminately it is possible that the quality and amount (479 656 training instances) were not suitable for these algorithms.

The feature vector extractor that generated feature vectors from the original training data plus its QLAS analysis did not translate absolutely all the original data. Having said that, it translated all that seemed intuitively relevant (see Table 3.1). The feature based methods did never come close to the classification performance of the entropy based methods.

The poorer performance can probably be attributed to two factors. The entropy based method used a more expensive analysis of the training data. It had also more information available (the feature vector extractor did not categorically translate all the information available into the feature vectors). This does not necessarily mean that including more information in the feature vector extractor would lead to better performance.

One of the stranger results in the feature vector experiment is the result of the IB1 algorithm (see Section 5.2.3). It performed worse than the expected result of random classification. This result can be seen as a permanently erroneous inductive bias. Three of the categories were classified slightly better than at random but one of the categories were consistently wrong. This lead to a average correctness that performed worse than the expected result of random classification.

6.3 Metric indexing

Using metric indexing gave a significant reduction in distance calculations. It did however come with a downside. Just finding the k closest objects and using these to do the classification gave a large degradation in classification performance.

Using metric indexing gave a significant reduction in distance calculations, but not as much as is desired. LAESA reduced the distance calculations by 45% – 47% and SSSTree by 22.75%, using $k = 10$. To improve the classification performance one could increase k, but that would come at the cost of more distance calculations. To increase the filtering performance one could decrease k, but that would come at the cost of classification performance.

Since the number of distance calculations at $k = 10$ is already high, it is not a good option to increase k. The classification performance at $k = 10$ was not that good either, much worse than the entropy experiment in Section 6.1.2. It is therefore not a good option to decrease k.

Increasing the number of pivots from 32 to 96 did not give a significant reduction in distance calculations either. This suggests that increasing the number of pivots further would not lead to a significant filtering performance.

The intrinsic dimensionality of the data set using normalized compression distance was estimated to **21.3**. This was estimated by using the mean and standard deviation of the 96 random pivot LAESA index. An intrinsic dimensionality of over 21 and the fact that Normalized Compression Distance is an approximated metric is probably the reason of why the metric indexing were not able to reduce the distance calculations further. The distance distribution histogram can be found in Figure 6.2.

6.4 Traditional information retrieval

Using a off-the-shelf search engine or more traditional information retrieval did not give very positive results. This is a method that usually works well in discriminating

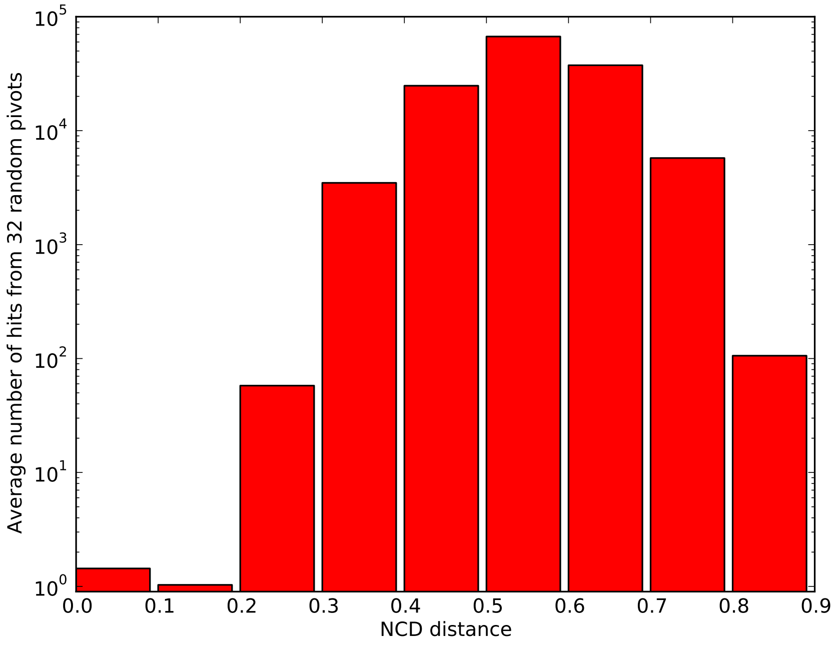


Figure 6.2: The distance distribution histogram of the 96 pivot LAESA index

between other types of documents. It is therefore fairly safe to conclude that the problem with this method is that the QLAS documents are not appropriate for this type of document ranking.

The entropy classification experiment in Section 6.1.2 gave a significant better result using the same data, at a much higher cost. The result of that experiment strongly indicate that the information is present in the QLAS data, but is it costly to do the analysis to find it. This experiment did not use any deep (expensive) analysis of data, just a typical bag-of-words model. Hence, the results are not so surprising.

It should be noted that the result of using Vespa is not comparable with the other results. Vespa has the option to not return any documents if a certain threshold is not met. The other experiments in this project did not have the option to do so. The classification result is therefore divided in five classes, the four previously used and "unclassified". The consequence if this is that the expected result of random classification is reduced from 25% to 20%.

The classification result using Vespa (see Section 5.3.1) displays a similar behavior to the direct zipping experiment (Section 3.2.1). The result suggest that there is a bias for each category and the category of the test instances have little influence on the classification. The *blog* category was categorized about 30% of the time, regardless of what test category was classified. The same holds for the other categories; *image* : 12%; *shopping* : 11%, *knowledge*: 40% and *unclassified*: 7%.

The result using Gensim (see Section 5.3.2) also have a biased tendency, although not comparable with Vespa. The *blog* category is overrepresented in all the test categories. But, unlike Vespa, each of the other categories are more often classified correctly, then incorrectly.

Chapter 7

Conclusion and further work

The most important conclusion of this project is that QLAS can in theory be used to do source classification. However, in its current form one have to use very expensive classification methods. The experiment that gave a highly statistical significant result is to computationally expensive to be used in a practical production system.

A big reason for why many of the results were not very positive can be attributed to the fact that QLAS was not designed for the purpose utilized in this project. Still, this does not mean that the QLAS project is useless in regards to source classification. This project used QLAS data "as is" with a standard QLAS installation. I believe that many of the components of QLAS could be used in a source classification system, e.g. the ability to place a query in a semantic ontology. This could be very valuable in a source classification system, as hypothesized in Section 1.1. But to be able to do this in a practical applicable system, the output from QLAS has to be built with a purpose for a system like that. Another possible modification could be to build a semantic ontology purpose-built for source classification.

7.1 Entropy classification

To be able to use this method in a practical system one would have to maintain a small data set of documents that are good at distinguishing between document categories. The training set would have to be reasonably small because the speed of classifying a document increases linearly with the size of the training set (the entire training corpus has to be compressed for each classification). This is a big limiting factor of this method, especially in a search engine where the goal is coping in an age of exponential information production [24]. Nonetheless, the result of the method was useful, in evaluating the value of the information in QLAS. It might be difficult to deploy in a commercial grade search engine, but the result it produced in this project were still valuable as a tool for data analysis.

In reading papers utilizing this method [22, 23, 6], it definitely seems to be a useful machine learning technique in many scenarios, e.g., when there is no obvious answer to a problem. This is the case in the "Tree of life" problem, where one wants to determine the relation of different organisms/genomes. This was called *exploratory machine learning* in [23]. Classifying web search queries in a practical production system on the other hand is not of this type. This is more similar to classic machine learning where one has positive examples in each class that the system is expected to classify. On the other hand, analyzing the value of information content in QLAS can be seen as exploratory machine learning or data mining and in this regard the experiment was a success.

7.2 Feature vector classification

The main limiting factor for this type of classification is the quality of the feature vectors in the training set. Using a naive feature vector extractor on the QLAS documents did not produce feature vectors that performed well. As stated in the beginning of this chapter, a modified QLAS or just using parts of QLAS could probably be used in some way to produce much better training data, and hence, feature vectors. This would require a re-engineering of QLAS and was out of scope for this project.

7.3 Traditional information retrieval

This experiment was a "shot in the dark". The experiment was done on the offhand chance that it would work. The expected results before doing the experiment were not highly optimistic. I still feel that the experiment was worth spending some time on. If it had a positive result it would have been very good news for Yahoo!. They could have used two production system, as is. This was however not the case.

7.4 Future work

This project did not result in a system that could be deployed as a production system. Several aspects needs further investigation.

Possible changes in QLAS

A investigation into the internal workings of QLAS is warranted. The entropy classification experiment showed that the necessary information is present, but not easily available. Based on this it would be interesting to do a project that evaluated possible changes in QLAS to improve this. It might even be possible to add source classification directly into QLAS.

Creating a semantic classification system from scratch

An alternative to modifying QLAS is to create a new semantic aware source classification system from scratch. This has the advantage that one can create a semantic ontology with this goal in mind. This would however be a very big task and much of the required work have probably already been done in creating QLAS.

Analyzing user behavior to produce a source classification system

A good source for learning the connection between queries and sources is the way users interact with a search engine. A simple model of this could be "n% of users refine a search query X to source Y, Y is a relevant source if n is higher than *threshold*". Other more elaborate methods probably exist. This user behavior analysis task is critical to generate data for a source classification system. It could either done in log analysis after the search is done or live as users interact with the system.

Coping with a steady stream of input

A search engine must cope with a steady stream of new information. A source classification system needs to handle this as well. Using semantics in the source classification might ease this by learning general connections between semantic classes but it should still be able to cope with a continuous stream of new inputs. The methods investigated in this project did not focus on this issue. This aspect have to be addressed to create a source classification system that can be used in a real world system.

Latent semantic indexing

The traditional information retrieval experiments did not give very positive results. One area one might explore further is if latent semantic indexing [12] or other advanced information retrieval could improve this performance.

Boosting

Boosting [33] is based in the question "can a set of *weak learners* create a single *strong learner*?" It could be interesting to see if combining several algorithms that were not able to produce very positive results in this project could be combined to create a stronger classifier.

7.5 Final thoughts

The most significant contribution of the project is the analysis of the feasibility of using QLAS in a source classification system. This produced valuable information. Unfortunately, the project did not result in any production grade algorithms or software.

Bibliography

[1] Internal yahoo! documentation. URL `http://yforge.corp.yahoo.com/ui/view/VespaPlatform`.

[2] Agnar Aamodt. Knowledge-intensive case-based reasoning in creek. In *In Proceedings of the Seventh European Conference on Case-Based Reasoning*, pages 1–15. Springer, 2004.

[3] David W. Aha and Dennis Kibler. Instance-based learning algorithms. In *Machine Learning*, pages 37–66, 1991.

[4] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, 1st edition, May 1999. ISBN 020139829X. URL `http://www.worldcat.org/isbn/020139829X`.

[5] Richard Bellman. *Dynamic Programming.* Dover Publications, March 2003. ISBN 0486428095. URL `http://www.worldcat.org/isbn/0486428095`.

[6] Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. Language trees and zipping, Dec 2001. URL `http://arxiv.org/abs/cond-mat/0108530`.

[7] Leo Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.

[8] Leo Breiman. Random forests. *Mach. Learn.*, 45:5–32, October 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL `http://portal.acm.org/citation.cfm?id=570181.570182`.

[9] Nieves Brisaboa, Oscar Pedreira, Diego Seco, Roberto Solar, and Roberto Uribe. Clustering-based similarity search in metric spaces with sparse spatial centers. In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrat, and Mária Bieliková, editors, *SOFSEM 2008: Theory and Practice of Computer Science*, volume 4910 of *Lecture Notes in Computer Science*, pages 186–197. Springer Berlin / Heidelberg, 2008.

[10] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001. URL `http://doi.acm.org/10.1145/502807.502808`.

[11] William W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[12] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407, 1990.

[13] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951 (Informational), May 1996. URL `http://www.ietf.org/rfc/rfc1951.txt`.

[14] Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning, 1994.

[15] Mark Hall and Eibe Frank. Combining naive bayes and decision tables. In *Proceedings of the 21st Florida Artificial Intelligence Society Conference (FLAIRS)*, pages 318–319. AAAI press, 2008.

[16] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, 2009. URL `http://dx.doi.org/10.1145/1656274.1656278`.

[17] Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction: with 200 full-color illustrations.* New York: Springer-Verlag, 2001.

[18] D. M. Hawkins. The Problem of Overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12, 2004. URL `http://pubs3.acs.org/acs/journals/doilookup?in_doi=10.1021/ci0342472`.

[19] Magnus Lie Hetland. The basic principles of metric indexing. In Carlos Coello, Satchidananda Dehuri, and Susmita Ghosh, editors, *Swarm Intelligence for Multi-objective Problems in Data Mining*, volume 242 of *Studies in Computational Intelligence*, pages 199–232. Springer Berlin / Heidelberg, 2009. URL `http://dx.doi.org/10.1007/978-3-642-03625-5_9`.

[20] Search Jack Menzel, GoogleGroup Product Manager. This week in search 12/18/09. *http://googleblog.blogspot.com/2009/12/this-week-in-search-121809.html*, 2009. URL `http://googleblog.blogspot.com/2009/12/this-week-in-search-121809.html`.

[21] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`.

[22] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the tenth ACM SIGKDD*

international conference on Knowledge discovery and data mining, KDD '04, pages 206–215, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: http://doi.acm.org/10.1145/1014052.1014077. URL `http://doi.acm.org/10.1145/1014052.1014077`.

[23] Ming Li, Xin Chen, Xin Li, Bin Ma, and P.M.B. Vitanyi. The similarity metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, Dec. 2004. ISSN 0018-9448. doi: 10.1109/TIT.2004.838101.

[24] Peter Lyman, Hal R. Varian, Peter Charles, Nathan Good, Laheem L. Jordan, and Joyojeet Pal. How much information? 2003, 2003. URL `http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/`.

[25] Luisa Micó, Jose Oncina, and Rafael C. Carrasco. A fast branch & bound nearest neighbour classifier in metric spaces. *Pattern Recogn. Lett.*, 17:731–739, June 1996. ISSN 0167-8655. doi: 10.1016/0167-8655(96)00032-3. URL `http://portal.acm.org/citation.cfm?id=230519.230528`.

[26] Oscar Pedreira and Nieves R. Brisaboa. Spatial selection of sparse pivots for similarity search in metric spaces. In *Proceedings of the 33rd conference on Current Trends in Theory and Practice of Computer Science*, SOFSEM '07, pages 434–445, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-69506-6. doi: http://dx.doi.org/10.1007/978-3-540-69507-3_37. URL `http://dx.doi.org/10.1007/978-3-540-69507-3_37`.

[27] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1022643204877.

[28] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

[29] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[30] E V Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recogn. Lett.*, 4:145–157, July 1986. ISSN 0167-8655. doi: 10.1016/0167-8655(86)90013-9. URL `http://portal.acm.org/citation.cfm?id=13010.13011`.

[31] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. ISBN 0137903952. URL `http://portal.acm.org/citation.cfm?id=773294`.

[32] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986. ISBN 0070544840.

[33] Robert E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5: 197–227, July 1990. ISSN 0885-6125. doi: 10.1023/A:1022648800760. URL `http://portal.acm.org/citation.cfm?id=83637.83645`.

[34] Julian Seward. bzip2 manual. *http://www.bzip.org/1.0.3/bzip2-manual-1.0.3.html*. URL `http://www.bzip.org/1.0.3/bzip2-manual-1.0.3.html`.

[35] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.

[36] Harald Weinreich, Hartmut Obendorf, Eelco Herder, and Matthias Mayer. Not quite the average: An empirical study of web use. *ACM Trans. Web*, 2:5:1–5:31, March 2008. ISSN 1559-1131. doi: http://doi.acm.org/10.1145/1326561.1326566. URL `http://doi.acm.org/10.1145/1326561.1326566`.

[37] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Comput.*, 8:1341–1390, October 1996. ISSN 0899-7667. doi: 10.1162/neco.1996.8.7.1341. URL `http://portal.acm.org/citation.cfm?id=1362127.1362128`.

[38] David H. Wolpert and William G. Macready. No free lunch theorems for search. Working Papers 95-02-010, Santa Fe Institute, February 1995. URL `http://ideas.repec.org/p/wop/safiwp/95-02-010.html`.

[39] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, 1(1):67–82, 1997.

[40] Harry Zhang. The Optimality of Naive Bayes. In Valerie Barr and Zdravko Markov, editors, *FLAIRS Conference*. AAAI Press, 2004. URL `http://www.cs.unb.ca/profs/hzhang/publications/FLAIRS04ZhangH.pdf`.

Appendix A

APPENDIX

A.1 QLAS

A.1.1 QLAS analysis example, query: "new york"

```
QLAS Analysis:
    decorations=
      [version]->"1.2.3"
    Interpretation:
      id=3
      Modification:
        id=1
        query=[new york]
        decorations=
            [domain]->[
              {
                "name":"domain_local",
                "prob":0.300649,
                "schema":"lm"
              },
              {
                "name":"domain_product",
                "prob":0.442650,
                "schema":"lm"
              }
            ]
            [score]->1.000000
      span=[0,3][new] id=8  class=token   referent=
      span=[4,4][york]    id=9  class=token   referent=
      span=[0,8][new york]  id=11 class=place_name    referent=
        decorations=
            [^taxonomy:place_category]->"/state"
            [conf]->0.982552
      segmentation()=[(new)(york)]
      decorations=
        [score]->0.991837
    Interpretation:
      id=4
      Modification:
        id=1
        query=[new york]
        decorations=
            [domain]->[
              {
                "name":"domain_local",
                "prob":0.300649,
                "schema":"lm"
              },
              {
                "name":"domain_product",
                "prob":0.442650,
                "schema":"lm"
              }
            ]
            [score]->1.000000
      span=[0,3][new] id=8  class=token   referent=
      span=[4,4][york]    id=9  class=token   referent=
      span=[0,8][new york]  id=14 class=place_name    referent=2459115
        decorations=
            [^taxonomy:place_category]->"/city"
            [woe_id]->2459115
      segmentation()=[(new)(york)]
      decorations=
        [score]->0.991837
    Interpretation:
      id=2
      Modification:
        id=1
        query=[new york]
        decorations=
            [domain]->[
              {
                "name":"domain_local",
                "prob":0.300649,
                "schema":"lm"
              },
              {
                "name":"domain_product",
                "prob":0.442650,
                "schema":"lm"
              }
            ]
            [score]->1.000000
      span=[0,3][new] id=8  class=token   referent=
      span=[4,4][york]    id=9  class=token   referent=
      span=[0,8][new york]  id=10 class=media_title
                                      referent=ymovies:1808741477
        decorations=
            [^taxonomy:media_category]->"/movie"
            [release_date]->"1996-10-30"
      segmentation()=[(new)(york)]
      decorations=
        [domain]->[
            {
              "name":"movie",
              "schema":"jabba",
              "slots":{
                "movie":[
                    10
                ]
              }
            }
        ]
        [score]->0.346093
    Interpretation:
      id=5
      Modification:
        id=1
        query=[new york]
        decorations=
            [domain]->[
              {
                "name":"domain_local",
                "prob":0.300649,
                "schema":"lm"
              },
              {
                "name":"domain_product",
                "prob":0.442650,
                "schema":"lm"
              }
            ]
            [score]->1.000000
      span=[0,3][new] id=8  class=token   referent=
      span=[4,4][york]    id=9  class=token   referent=
      span=[4,4][york]    id=12 class=brand_name    referent=
        decorations=
            [^taxonomy:brand_type]->"/manufacturer"
      segmentation()=(new)[(york)]
      decorations=
        [score]->0.161142
    Interpretation:
      id=7
      Modification:
        id=1
        query=[new york]
        decorations=
            [domain]->[
              {
                "name":"domain_local",
                "prob":0.300649,
                "schema":"lm"
              },
              {
                "name":"domain_product",
                "prob":0.442650,
                "schema":"lm"
              }
            ]
            [score]->1.000000
      span=[0,3][new] id=8  class=token   referent=
      span=[4,4][york]    id=9  class=token   referent=
      segmentation()=(new)(york)
      decorations=
        [domain]->[
            {
              "implicit":true,
              "name":"image",
              "schema":"jabba",
              "slots":{

              }
            }
```

```
        ]
        [score]->0.086218
    Interpretation:
      id=6
      Modification:
        id=1
        query=[new york]
        decorations=
            [domain]->[
              {
                "name":"domain_local",
                "prob":0.300649,
                "schema":"lm"
              },
              {
                "name":"domain_product",
                "prob":0.442650,
                "schema":"lm"
              }
            ]
            [score]->1.000000
      span=[0,3][new] id=8  class=token   referent=
      span=[4,4][york]    id=9  class=token   referent=
      span=[4,4][york]    id=13 class=organization_name referent=
      segmentation()=(new)[(york)]
      decorations=
        [score]->0.064981


analysis time:  43.851 ms
total:  43.851 ms
```

A.1.2 QLAS Domains

This section contains information about the domains a QLAS analysis can contain.

Language Model

Language Model (LM) domains are created when an LM classifier decides the query is in a class with a certain confidence. The ”name” field will contain a classification of the modification (one of “domain_autos”, “domain_local”, “domain_product”, “domain_travel”), and there will be an additional ”prob” field in the range [0,1] which specifies some confidence in the classification.

domain name	**interpretation meaning**	**example queries**	**other fields**
event	refers to some event	-	
image	has image intent	-	implicit
local	has local intent	plumbers in sunnyvale	implicit
map	has map intent	sunnyvale map	implicit
movie	contains or is related to a movie or movies generally	the matrix, shrek 2 showtimes in sunnyvale	task
music	-	-	
product	contains or is related to a produ ct or product category	mp3 players, apple ipod touch 80gb	task
reference	specific reference task	synonym of obfuscated, how many feet in 5 meters	task, dimension
travel	related to travel or has travel intent	paris vacations, delta flight 123	task
weather	has weather intent	sunnyvale weather	

Table A.1: Different jabba domains

An example of a LM interpretation can be found in Figure A.1.

Jabba

Jabba domains are created when an interpretation matches a Jabba rule. The schema field will be jabba, and there will be another slots field whose value is itself an object. Jabba rules may also set arbitrary fields in the domain object.

An example of a QLAS interpretation labeled with a jabba domain can be found in Figure A.2.

A full list of all the posible domains can be found in Table A.1. Each domain can in addition condain extra fields, they are all listed in Table A.2, A.3.

Coexistence

Coexistence domains are created when a simple coexistence tagger detects two spans of certain classes together.

```
...
Interpretation:
   id=19
   Modification:
      id=1
      query=[restaurants in sunnyvale]
      ...
   span=[0,11][restaurants]   id=2   class=token   referent=
   span=[12,2][in]   id=3   class=token   referent=
   span=[15,9][sunnyvale]   id=4   class=token   referent=
   span=[0,11][restaurants]   id=5   class=business   referent=
      decorations=
         [seq_prob]->0.912692
         [tag_prob]->1.000000
   span=[15,9][sunnyvale]   id=10   class=place_name   referent=2502265
      decorations=
         [^taxonomy:place_category]->"/city"
         [seq_prob]->0.912692
         [tag_prob]->1.000000
         [woe_id]->2502265
   segmentation()=[(restaurants)](in)[(sunnyvale)]
   decorations=
      [domain]->[
         {
            "name":"domain_local",
            "schema":"lm",
            "prob":0.808629
         },
         {
            "name":"domain_travel",
            "schema":"lm",
            "prob":0.638892
         }
      ]
      [score]->3.122428
...
```

Figure A.1: LM interpretation

common slots		
slot name	*meaning*	*examples*
directive	directive spans which indicated the rule match	sunnyvale map, paris vacations, shrek 2 showtimes in sunnyvale
location	entire location	plumbers in sunnyvale ca 94089
common fields		
”implicit” : true	rule match is based on implicit meaning of the query or parts	pizza (triggers local)
local slots		
slot name	*meaning*	*example*
business	business	plumbers in sunnyvale
movie slots		
slot name	*meaning*	*example*
movie	movie entity that is the object of the query	shrek 2 showtimes in sunnyvale
movie fields		
no task field	generally movie-related	the matrix
”task” : ”review”	interpretation has review intent for specific movie	movie reviews for avatar
”task” : ”showtime”	interpretation has showtime intent for specific movie	shrek 2 showtimes in sunnyvale
”task” : ”trailer”	interpretation has movie trailer intent for specific movie	waterworld trailers
music slots		
slot name	*meaning*	*example*
slot name	meaning	example
artist	artist entity that is the object of query	-
product slots		
slot name	*meaning*	*example*
attributes	attributes associated with product	black ipod 80gb
category	product category	mp3 players
item	product item	apple ipod
product fields		
no task field	interpretation is product item or category	mp3 players
”task” : ”review”	interpretation has review intent for specific product	apple ipod touch 80gb

Table A.2: Jabba fields, Table 1

```
[domain]->[
  {
    "name":"movie",
    "schema":"jabba",
    "slots":{
      "movie":[
          10
      ]
    }
  }
]
```

Figure A.2: Example of a jabba domain in a QLAS analysis

reference slots		
slot name	*meaning*	*example*
slot name	meaning	example
object	main object of reference request	synonym of obfuscated
quantity	quantity to be converted	how many feet in 5 meters
from	conversion source	how many feet in 5 meters
to	conversion destination	how many feet in 5 meters
reference fields		
”task” : ”thesaurus”	thesaurus intent	synonym of obfuscated
”task” : ”conversion”	info on a specific flight	how many feet in 5 meters
”dimension” : ”measurement”	denotes conversion task is for measurements	how many feet in 5 meters
travel slots		
slot name	*meaning*	*example*
origin	travel origin	flight from sfo to jfk
destination	travel destination	flight from chicago to paris
flight_number	flight number only, not including airline	delta flight 123
travel fields		
no task field	-	-
”task” : ”guide”	info about a travel destination	paris vacations
”task” : ”flight”	info on a specific flight or flights	delta flight 123
”task” : ”hotel”	info about hotel/motel/b	b & 5 star hotels in paris

Table A.3: Jabba fields, Table 2

Table A.4: QLAS class taxonomy, table 1

type	class name	definition	"colon"-equivalent	example terms	attribute name	attribute type: value
syntactic	token	Created by tokenizer		If an is attribute does not exist, it is assumed to be false.	is stopword	boolean
					is numeric	boolean
					is quoted	boolean
					modifiers	string, e.g. "+"
no classification	compound	created for quoted phrases	other			
proper noun	person name	Names of people	person name: *	Britney Spears, Conan the Barbarian	t̂axonomy: notability category	string: taxonomy node e.g. /notable/arts media/musician
	media title	Titles of media works	media title: *	Rush Hour 3, Smells Like Teen Spirit	t̂axonomy: media category	string: taxonomy node e.g. /movie
					release date	string e.g. 1999-03-31
	organization name	Names of organizations	organization name: *	Santa Clara Marriott, Nine Inch Nails	t̂axonomy: industry	string: taxonomy node e.g. /travel/lodging
					is chain	boolean
					is company	boolean
	brand name	Brand names (any part of a brand name).	brand name: *	Nike, Playstation	t̂axonomy: brand type	string: taxonomy node e.g. /manufacturer
					t̂axonomy: industry	string: taxonomy node e.g. /technology electronics
	measurement name	Symbols and names	measurement name	pint, mile	-	

Table A.5: QLAS class taxonomy, table 2

	currency name	Long and short names	currency name	USD, U.S. dollar		
	disease and condition	-	health: disease and condition	sinusitis, nervous system disorders	-	
	horoscope sign	-	horoscope sign	aries, gemini	-	
	place name	-	place: *	California, London, France, Santa Clara County, Manhattan	ˆtaxonomy: place category	string: taxonomy node e.g. /airport
					woe id	string: WOEID e.g. 2502265 (note: the type is string not integer)
	event name	-	event name: *	World War 2, Christmas	ˆtaxonomy: event category	string: taxonomy node e.g. /holiday
common noun	occasion	Generic terms such as birthday or anniversary should go in this category.	occasion	wedding, birthday	-	
	business	For types of businesses, not names of businesses.	businesscategory: *	electronics repair, plumber, salon, boot camp	ˆtaxonomy: industry	string: taxonomy node e.g. /automotive
	beverage	Common beverage terms	category: food	cocktail	-	
	dish	Specific food dishes	category: food	trout amandine, paella	-	
	drink	Specific drinks	martini, bloody mary	-		
	food	Common food terms (note: terms like recipe are directives)	category: food	dessert, soup, appetizer, frozen dinner	-	
	foodingredient	Ingredients of food dishes	attribute: food: ingredient	chicken, chocolate	-	

Table A.6: QLAS class taxonomy, table 3

	product	Includes generic product terms	productcategory: *	mp3 player, dvd, cars	̂taxonomy: industry	string: taxonomy node e.g. /automotive
	treatment	-	health: treatment	chemotherapy, plastic surgery	-	
adjective	color	Any color [Question: what about patterns like polka dot, striped, plaid?]	attribute: color	pink, ecru, beige, black	-	
	cookingmethod	Method of cooking	attribute: food: methodofcooking	fried, sauteed, slow cooked	-	
	cuisine	Food cuisine	attribute: food: cuisine	chinese, indian, continental	-	
	diet	Food diet	attribute: food: diet	low fat, vegan, vegetarian	-	
	foodtaste	Food taste	attribute: food: taste	spicy	-	
code	airlinecode	-	code: airlinecode	UA, SW	-	
	companyticker	-	code: companyticker	MBH, YHOO	-	
	domain	Domain names	code: domain	myspace.com	-	
	emailaddress	Includes malformed email addresses (even if an email is not presented correctly, the intent is there)	code: emailaddress	johnsmith@yahoo.com	-	
	fedextracking	Fedex package tracking number	code: fedex	-	-	
	filename	A name of a file or a file type	code: filename	blaster.exe, mscomct2.ocx, jpg, avi	-	

Table A.7: QLAS class taxonomy, table 4

	isbn	-	code:isbn	-	-	
	ncode	US aircraft tail number	-none-	-	-	
	phonenumber	Phone numbers with or without area code, country code, international dialing code, etc.	code:phonenumber	800 555-5555, (408) 555-5555	-	
	upc	Universal product code	code:upc	-	-	
	upstracking	UPS package tracking number	code:ups	-	-	
	uspstracking	USPS package tracking number	code:usps	-	-	
	url	For specific url-intent terms.	code:url	www.myspace.com, http://www.facebook.com	-	
	vin	Vehicle identification number	code:vin	-	-	

A.1.3 QLAS Taxonomy

A.2 Experiment output

The results of the experiments are summarized in Chapter 5. This section gives the full output of the experiments.

A.2.1 Naive Bayes

```
Naive Bayes Classifier




Time taken to build model: 3.61 seconds
Time taken to test model on training data: 7.11 seconds

=== Error on training data ===

Correctly Classified Instances      175572               36.6037 %
Incorrectly Classified Instances    304084               63.3963 %
Kappa statistic                          0.1557
Mean absolute error                      0.3374
Root mean squared error                  0.4376
Relative absolute error                 89.9936 %
Root relative squared error            101.0623 %
Total Number of Instances           479656


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.047     0.019      0.449     0.047     0.086      0.596    blog
  0.322     0.238      0.309     0.322     0.315      0.576    image
  0.337     0.196      0.38      0.337     0.357      0.634    knowledge
  0.765     0.392      0.385     0.765     0.513      0.738    shopping
  0.366     0.21       0.381     0.366     0.318      0.636


=== Confusion Matrix ===

     a     b     c     d   <-- classified as
  5588 31457 34319 46600 |     a = blog
  2140 38376 24574 54246 |     b = image
  4010 37957 42394 41385 |     c = knowledge
   698 16348 10350 89214 |     d = shopping


=== Error on test data ===

Correctly Classified Instances       19111               35.9865 %
Incorrectly Classified Instances     33995               64.0135 %
Kappa statistic                          0.1471
Mean absolute error                      0.3399
Root mean squared error                  0.441
Relative absolute error                 90.6762 %
Root relative squared error            101.8523 %
Total Number of Instances            53106


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.044     0.026      0.352     0.044     0.078      0.574    blog
  0.316     0.234      0.312     0.316     0.314      0.578    image
  0.347     0.197      0.384     0.347     0.365      0.637    knowledge
  0.737     0.396      0.374     0.737     0.496      0.717    shopping
  0.36      0.213      0.356     0.36      0.314      0.626


=== Confusion Matrix ===

    a    b    c    d   <-- classified as
  566 3295 3694 5432 |    a = blog
  289 4220 2774 6082 |    b = image
  525 4094 4807 4422 |    c = knowledge
  226 1910 1252 9518 |    d = shopping
```

A.2.2 DTNB

```
=== Run information ===

Scheme:       weka.classifiers.rules.DTNB -X 1
Relation:     training
Instances:    479656
```

```
Attributes:   10
              class
              lang
              prob_a
              prob_b
              prob_c
              qlas_class
              qlas_class_prob
              qlas_taxonomy
              qlas_taxonomy_score
              qlas_taxonomy_dec
Test mode:    user supplied test set:  size unknown (reading incrementally)

=== Classifier model (full training set) ===

Decision Table:

Number of training instances: 479656
Number of Rules : 66870
Non matches covered by Majority class.
Evaluation (for feature selection): CV (leave one out)
Feature set: 3,5,6,7,1

Time taken to build model: 360.26 seconds

=== Evaluation on test set ===
=== Summary ===

Correctly Classified Instances       21298               40.1047 %
Incorrectly Classified Instances     31808               59.8953 %
Kappa statistic                          0.202
Mean absolute error                      0.3171
Root mean squared error                  0.4496
Relative absolute error                 84.5708 %
Root relative squared error            103.8546 %
Total Number of Instances            53106

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.284     0.15       0.38      0.284     0.325      0.62     blog
  0.317     0.184      0.367     0.317     0.34       0.612    image
  0.373     0.194      0.403     0.373     0.388      0.64     knowledge
  0.636     0.27       0.431     0.636     0.514      0.744    shopping
  0.401     0.199      0.395     0.401     0.391      0.653
```

```
=== Confusion Matrix ===

    a    b    c    d   <-- classified as
 3690 2444 3208 3645 |    a = blog
 2033 4236 2992 4104 |    b = image
 2748 2847 5163 3090 |    c = knowledge
 1238 2024 1435 8209 |    d = shopping
```

A.2.3 IB1

```
IB1 classifier

Time taken to build model: 4.95 seconds
Time taken to test model on training data: 70340.82 seconds

=== Error on training data ===

Correctly Classified Instances      291395               60.7508 %
Incorrectly Classified Instances    188261               39.2492 %
Kappa statistic                          0.477
Mean absolute error                      0.1962
Root mean squared error                  0.443
Relative absolute error                 52.3471 %
Root relative squared error            102.3202 %
Total Number of Instances           479656


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.8       0.281      0.481     0.8       0.601      ?        blog
  0.599     0.122      0.619     0.599     0.609      ?        image
  0.578     0.077      0.728     0.578     0.644      ?        knowledge
  0.454     0.043      0.773     0.454     0.572      ?        shopping
  0.608     0.13       0.651     0.608     0.607      0


=== Confusion Matrix ===

     a     b     c     d   <-- classified as
 94359  9479  9291  4835 |     a = blog
 33545 71431  8515  5845 |     b = image
 33392 14887 72626  4841 |     c = knowledge
 34707 19532  9392 52979 |     d = shopping


=== Error on test data ===

Correctly Classified Instances       18049               33.9867 %
Incorrectly Classified Instances     35057               66.0133 %
Kappa statistic                          0.1205
Mean absolute error                      0.3301
Root mean squared error                  0.5745
Relative absolute error                 88.041  %
Root relative squared error            132.6947 %
Total Number of Instances            53106


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.5       0.369      0.305     0.5        0.379      0.566   blog
  0.302     0.214      0.322     0.302      0.312      0.544   image
  0.294     0.179      0.367     0.294      0.326      0.557   knowledge
  0.267     0.117      0.423     0.267      0.327      0.575   shopping
  0.34      0.219      0.354     0.34       0.336      0.56


=== Confusion Matrix ===

    a    b    c    d   <-- classified as
 6499 2430 2672 1386 |    a = blog
 4830 4037 2577 1921 |    b = image
 5361 3024 4071 1392 |    c = knowledge
 4622 3056 1786 3442 |    d = shopping
```

A.2.4 JRip

```
JRIP rules:
===========


Number of Rules : 68
```

```
Time taken to build model: 1785.83 seconds
Time taken to test model on training data: 3.95 seconds

=== Error on training data ===

Correctly Classified Instances      156281               32.5819 %
Incorrectly Classified Instances    323375               67.4181 %
Kappa statistic                          0.0895
Mean absolute error                      0.3621
Root mean squared error                  0.4255
Relative absolute error                 96.5934 %
Root relative squared error             98.2819 %
Total Number of Instances           479656


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.051     0.005      0.78      0.051     0.096      0.562    blog
  0.026     0.005      0.614     0.026     0.049      0.531    image
  0.929     0.833      0.284     0.929     0.435      0.549    knowledge
  0.26      0.069      0.549     0.26      0.353      0.608    shopping
  0.326     0.238      0.553     0.326     0.236      0.562


=== Confusion Matrix ===

      a      b      c      d   <-- classified as
   6035    810 104435   6684 |      a = blog
    472   3057 104390  11417 |      b = image
   1209    838 116821   6878 |      c = knowledge
     19    271  85952  30368 |      d = shopping


=== Error on test data ===

Correctly Classified Instances       16513               31.0944 %
Incorrectly Classified Instances     36593               68.9056 %
Kappa statistic                          0.0709
Mean absolute error                      0.3651
Root mean squared error                  0.4286
Relative absolute error                 97.3761 %
Root relative squared error             98.9917 %
Total Number of Instances            53106
```

```
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.028     0.009      0.494     0.028     0.052      0.547    blog
  0.021     0.005      0.591     0.021     0.04       0.525    image
  0.912     0.841      0.277     0.912     0.424      0.536    knowledge
  0.252     0.075      0.519     0.252     0.339      0.596    shopping
  0.311     0.241      0.468     0.311     0.216      0.551


=== Confusion Matrix ===

     a     b     c     d   <-- classified as
   358    67 11758   804 |     a = blog
    54   279 11681  1351 |     b = image
   289    80 12630   849 |     c = knowledge
    23    46  9591  3246 |     d = shopping
```

A.2.5 J48

```
Time taken to build model: 1606.26 seconds
Time taken to test model on training data: 3.54 seconds

=== Error on training data ===

Correctly Classified Instances      298257               62.1814 %
Incorrectly Classified Instances    181399               37.8186 %
Kappa statistic                          0.4961
Mean absolute error                      0.2464
Root mean squared error                  0.3485
Relative absolute error                 65.7339 %
Root relative squared error             80.4953 %
Total Number of Instances           479656


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.542     0.105      0.628     0.542     0.582      0.842    blog
  0.507     0.089      0.653     0.507     0.571      0.832    image
  0.616     0.109      0.667     0.616     0.64       0.858    knowledge
  0.827     0.2        0.57      0.827     0.675      0.899    shopping
  0.622     0.125      0.63      0.622     0.617      0.857


=== Confusion Matrix ===

     a     b     c     d   <-- classified as
 63932 12440 18568 23024 |     a = blog
 13691 60496 15363 29786 |     b = image
 16245 12253 77435 19813 |     c = knowledge
  7954  7468  4794 96394 |     d = shopping


=== Error on test data ===

Correctly Classified Instances       21283               40.0765 %
Incorrectly Classified Instances     31823               59.9235 %
Kappa statistic                          0.2019
Mean absolute error                      0.3269
Root mean squared error                  0.4486
Relative absolute error                 87.1897 %
Root relative squared error            103.6189 %
Total Number of Instances            53106


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.331     0.184      0.368     0.331     0.349      0.626    blog
  0.277     0.172      0.352     0.277     0.31       0.582    image
  0.362     0.181      0.413     0.362     0.386      0.615    knowledge
  0.64      0.261      0.441     0.64      0.522      0.748    shopping
  0.401     0.199      0.393     0.401     0.391      0.642


=== Confusion Matrix ===

    a    b    c    d   <-- classified as
 4304 2320 3049 3314 |    a = blog
 2522 3702 2851 4290 |    b = image
 3186 2760 5019 2883 |    c = knowledge
 1683 1743 1222 8258 |    d = shopping
```

A.2.6 Random Forest

```
Random forest of 10 trees, each constructed while considering 4 random features.
Out of bag error: 0.3837



Time taken to build model: 3499.39 seconds
Time taken to test model on training data: 570.56 seconds

=== Error on training data ===

Correctly Classified Instances      380745               79.3788 %
Incorrectly Classified Instances     98911               20.6212 %
Kappa statistic                          0.7252
Mean absolute error                      0.1828
Root mean squared error                  0.2776
Relative absolute error                 48.7638 %
Root relative squared error             64.108  %
Total Number of Instances           479656


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.759     0.062      0.8       0.759     0.779      0.955    blog
  0.753     0.05       0.833     0.753     0.791      0.955    image
  0.785     0.043      0.866     0.785     0.823      0.96     knowledge
  0.88      0.119      0.704     0.88      0.782      0.961    shopping
  0.794     0.068      0.802     0.794     0.794      0.958


=== Confusion Matrix ===

      a      b      c      d   <-- classified as
  89574   7006   6706  14678 |      a = blog
   7761  89884   5505  16186 |      b = image
   8707   6043  98722  12274 |      c = knowledge
   5923   4993   3129 102565 |      d = shopping


=== Error on test data ===

Correctly Classified Instances       22752               42.8426 %
```

```
Incorrectly Classified Instances     30354               57.1574 %
Kappa statistic                          0.2383
Mean absolute error                      0.3179
Root mean squared error                  0.4213
Relative absolute error                 84.8034 %
Root relative squared error             97.3157 %
Total Number of Instances            53106


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.329     0.161      0.399     0.329     0.361      0.659    blog
  0.32      0.163      0.398     0.32      0.355      0.631    image
  0.423     0.199      0.428     0.423     0.425      0.673    knowledge
  0.647     0.239      0.465     0.647     0.541      0.792    shopping
  0.428     0.19       0.422     0.428     0.42       0.688


=== Confusion Matrix ===

    a    b    c    d   <-- classified as
 4274 2161 3597 2955 |    a = blog
 2126 4280 2995 3964 |    b = image
 2870 2427 5853 2698 |    c = knowledge
 1447 1882 1232 8345 |    d = shopping
```

A.2.7 Vespa output

```
<result total-hit-count="84974">
  <hit relevancy="0.14760351294762875" source="qlas">
    <field name="category">knowledge</field>
    <field name="documentid">doc:qlas:1291956383:</field>
  </hit>
  <hit relevancy="0.12751657434115707" source="qlas">
    <field name="category">shopping</field>
    <field name="documentid">doc:qlas:-2109994265:</field>
  </hit>
  <hit relevancy="0.12479737839875495" source="qlas">
    <field name="category">blog</field>
    <field name="documentid">doc:qlas:-20109161:</field>
  </hit>
  <hit relevancy="0.12441472907126407" source="qlas">
    <field name="category">knowledge</field>
    <field name="documentid">doc:qlas:-1948869789:</field>
  </hit>
  <hit relevancy="0.1225546367713739" source="qlas">
```

```
    <field name="category">knowledge</field>
    <field name="documentid">doc:qlas:1883930451:</field>
  </hit>
  <hit relevancy="0.12204705399273562" source="qlas">
    <field name="category">image</field>
    <field name="documentid">doc:qlas:795608533:</field>
  </hit>
  <hit relevancy="0.12180796420934385" source="qlas">
    <field name="category">knowledge</field>
    <field name="documentid">doc:qlas:731226415:</field>
  </hit>
  <hit relevancy="0.11959180646868277" source="qlas">
    <field name="category">blog</field>
    <field name="documentid">doc:qlas:-2057640103:</field>
  </hit>
  <hit relevancy="0.11893421879975129" source="qlas">
    <field name="category">shopping</field>
    <field name="documentid">doc:qlas:-203011059:</field>
  </hit>
  <hit relevancy="0.11866847512874994" source="qlas">
    <field name="category">knowledge</field>
    <field name="documentid">doc:qlas:-798616389:</field>
  </hit>
</result>
```