University of Ca' Foscari

Master Thesis

Discovering Single-Query Tasks from Search Engine Logs

Author:
Farzad Vaziri

Supervisor:
Prof. Salvatore Orlando

October of 2014

Acknowledgment

I am so glad that we did this work but it is obvious that without having two great people beside me I could not do it at all, Prof. Salvatore Orlando and Dr. Gabriele Tolome. I owe them so much because at the beginning they help me so patiently when I was new in this topic and their kindly help was great encouragement for me to keep going ahead and finishing this work. I think the only person who was beside me for solving all the problems as an international student was Prof. Orlando. And, he finished his great work by helping me and guiding me to do this work. Beside him, Garbriele was always with me to do this work. Thank you so much for your kindness. I am so appreciated.

Abstract

Nowadays web search companies record all the query logs that different users submit for searching information on search engine. At first glance may be it does not look so interesting keeping and saving query logs which most of them even do not have an structure, but in-depth studies on query logs that have done till now show that they are not just some words and absolutely they could be useful for some other goals. However, query logs can not only analyzed to find out just some simple information about users' activities but also we can use data mining methods to extract knowledge to develop new search-related applications. For example query suggestions, finding better results for queries, recommendation systems and etc. As web queries mostly are short and have a simple structure, it will be difficult to identify the find the exact information needed by user. Then, some studies have done for dividing queries according to different measures to make them more clearly about the goal which they are done. Session detection was one of the concepts is used for web search query identifications. For query session detection different measure have used. Some of them just use time as a value for segmentation, some of them use time Main target of session detection is the finding typical patterns the users follow in their search processes, and, based on these patterns, to develop search support tools such as smart query suggestions learned from the reformulations of other users. Then, concept of search mission comes out which is connecting different search session with same information need which actually defines as search mission task. Among all the mission that have done in a session, there are mission which are not related to the information need which mission is done for it. And, they could decrees the performance the analyzing and mining the queries log. Finding these queries and eliminating them in one hand could be good factor for improving all result and on the other hand is hard because of enormous size of queries.

Contents

1 Introduction **5**
1.1 Search Engine Query Logs 5
1.2 Machine learning and data mining 7
1.3 Knowledge Discovery in Databases 7
1.3.1 Pre-processing 8
1.4 Weka 11
1.4.1 History of the Weka Project 11

2 State of The Art **13**
2.1 Query Log Mining 13
2.1.1 Query log exploitation 13
2.2 Search Session Definition 15
2.3 Analyzing behavior of user in a search session 16
2.4 Query Classification and Session detection methods 17
2.4.1 Temporal clues for session boundary detection 17
2.4.2 Lexical clues for session boundary detection 17
2.4.3 Machine-learning methods to combine temporal and lexical clues 17
2.5 Search mission detection 18
2.6 Singletone mission 18

3 Problem Statement **20**
3.1 Background and Notation 20
3.2 Problem Definition 21
3.3 Features Extraction 22
3.3.1 Non-Contextual Features 23
3.3.2 Contextual (Within-Session) Features 24
3.3.3 Semantic Contextual (Within-Session) Features 27

4 Modeling and Evaluation of models **30**
4.1 Data mining task (Classification) 30

4.2 Evaluating learned model 30
4.3 Commonly-accepted performance evaluation measures 31
4.3.1 Why Accuracy and Error Rate are Inadequate Performance Measures for Imbalanced Data Sets 31
4.3.2 What is a good Performance measure for our data set 33
4.3.3 Area under the curve 34
4.4 Training and Testing 35
4.4.1 ZeroR Algorithm 37
4.5 Most Influential Data Mining Algorithms 38
4.5.1 Bayesian network 38
4.5.2 LogitBoost 39
4.5.3 Threshold Selector with Logistic Algorithm 40
4.5.4 C4.5 - Decision trees 42
4.5.5 ADTree 44
4.6 Imbalanced data and skewed distributions 44
4.6.1 What is a imbalanced data set 44
4.7 What is the solution 46
4.8 More Experiments 47
4.8.1 Options available in Weka or re-sampling 47
4.8.2 Modelling with new resampled traning data set 48
4.8.3 Logistic Regression with new data set 48
4.8.4 Bagging Algorithm with new data set 49
4.8.5 LoogitBoost Algorithm with new train set 50
4.9 Under-sampling 52
4.10 Cost sensitive classification in output 52
4.10.1 Logestic Algorithm withwith cost sensitive classification 53
4.10.2 Bagging and LogitBoost Algorithms with cost sensitive classification 53

5 Conclusion and Future work **56**
5.1 Conclusion 56
5.2 Future work 57

Chapter 1

Introduction

1.1 Search Engine Query Logs

At these days search engines are getting to be a part of daily life of majority of people. People use search engine for more or less every things. The queries that people submit can be either a “Navigational query” — where the immediate intent is to reach a particular site, an “Informational query” — where the intent is to acquire some information assumed to be present on one or more web pages, a “Transactional queries” — where the intent is to perform some web-mediated activity. And, this leads that a lot of beneficial and non-beneficial options open for search engines. So, all search engines try to keep their portion of this opened gate for making more money. One of the resources which can be used for more study and analysis in search engine queries are Query Logs that recorded and actually submitted by users during their search period. Till now, different works have done by researchers that most of them are about classification and categorizing queries according to their information needs and tasks. Most known filed which is still has a lot of potentiality to grow and getting better is Session Detection which is about classification which classifies queries by theirs submitted time or newly done lexical features. But, there is a new filed about queries which goes to be a topic in query analysis is “Mission and Task Detection”. Bunch of works has done about this newly introduced works and still there are some aspect that missing about this work. A session contains one or more missions. These missions can be dependent to the information need or can be just a single task that have done within a session. Finding end extracting these single task queries automatically seems interesting enough to start a new topic and research. So, we planned to use data mining knowledge discovery algorithm for make some models and predictors for finding a pattern and making in popular for the other data set of query logs. Therefore, we should have been able to define some features and attributes to use them as input data set to our machine learning and data mining procedure. We had a collection of recorded queries with some information about each query. First step was defining and extracting as much as possible feature for each query which is submitted. For the preprocessing and data selection, we used a corpus of 6381 queries annotated with logical session and mission information with number of click per each query and the clicked links. In feature extracting section we tried to extract as much as available features for each query which were lexical and semantic all together. For, data mining task part we used the Weka software developed by "Wikato university of New Zealand" for Data mining and machine learning purposes. As this software has an adequate collection of algorithms we tried to do more suitable algorithms according to the types of our data which were numeric. After extracting the features and making a data set of all features with their value, we had another problem and it was the skewed and imbalanced of class distribution which leaded inaccurate results for our prediction model. By using over-sampling, under-sampling and cost matrix during our model constriction we solved this problem. Because of the type of features and the imbalance distribution of class attribute evaluating outputs and results based on common measures were not possible for our work, then we focused on "Kappa Value" plus some new visual methods that Weka makes them available for data mining analysis help so much easy understanding the results and the performance of made models like "ROC Curve" and "Threshold Curves".

Discovering single task queries absolutely would be useful for improving suggestion tools for offering more related results for the users and it means finding needed information in the minimum time for user. Since, there is a big competition among the biggest search engines for having been used by the more users, even one small step like this will help them in gathering more user. As we mentioned in Feature definition section, till now we worked on some lexical contextual and non-contextual features plus some semantic features which main one was "Wikipedia Similarity". But, as we know there are so many other semantic measure which can be defined for evaluating the semantic similarity of two text, our plan is applying them for queries and using them as new feature. There will be some boundaries because of the shortness of queries but still there are more options to use the semantic similarity measure for our feature work and making our classification model more and more precise about finding the single task queries.

1.2 Machine learning and data mining

Machine learning and data mining are two confusing field of science, since they employ the equal methods and overlap in many steps that they do. They can be roughly defined as follows:

✓Machine learning focuses on prediction, based on known properties learned from the training data.

✓Data mining focuses on the discovery of (previously) unknown properties in the data. This is the analysis step of Knowledge Discovery in Databases.

They have overlapping in many procedures: data mining employs many machine learning methods, but usually with a bit different target in mind. On the other hand, machine learning uses also data mining techniques as "unsupervised learning" or as a preprocessing step to make better learner accuracy. The confusion between these two field communities comes from the basic assumptions they work with: in machine learning, performance is usually assessed according to the ability to reproduce known knowledge, whenever in Knowledge Discovery and Data Mining (KDD) the key task is the discovery of previously unknown knowledge. Evaluated with respect to known knowledge, an uninformed (unsupervised) method will easily be outperformed by supervised methods, while in a typical KDD task, supervised methods cannot be used due to the unavailability of training data.

Data mining which is the analysis part of the "Knowledge Discovery in Databases" is a subfield of computer science that tries to discover patterns in large data sets involving at the intersection of artificial intelligence, machine learning, statistics, and database systems.[2]

As mentioned above, purpose of data mining procedure is to extract knowledge and information from a data set and change the form into an understandable shape for the future uses.

1.3 Knowledge Discovery in Databases

The Knowledge Discovery in Databases (KDD) process is defined with the following steps:

1. Selection
2. Pre-processing

3. Transformation

4. Data Mining

5. Interpretation/Evaluation.

or a simplified process such as (1) pre-processing, (2) data mining, and (3) results validation.

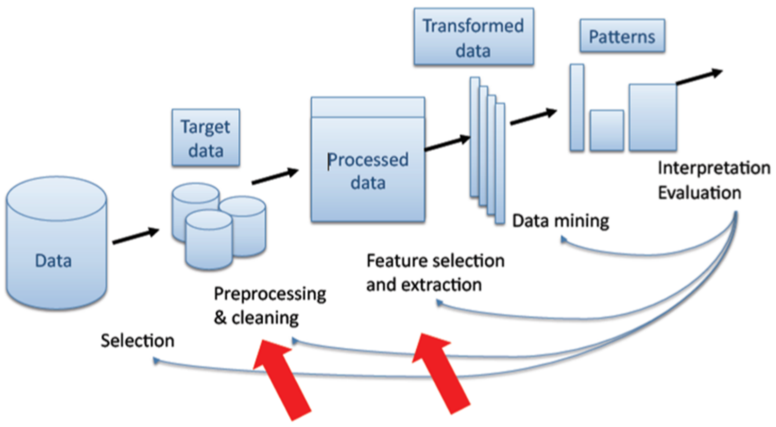


Figure 1.1: Knowledge Discovery in Database processes

1.3.1 Pre-processing

We have to assemble data set before using the data mining algorithms. Since data mining can only uncover patterns that practically exist in the data. The data set have to be large enough to contain these patterns. While remaining short enough to be mined with the algorithms within an acceptable time. Pre-processing is required to analyze the multivariate data sets before data mining. The target set is then cleaned. Data cleaning eliminates the observations that have noise and those with missing data.

Data transformation

In this step, producing of better data for the data mining is prepared and done. Techniques which are used is about dimension reduction (as feature selection and extracting the instances), and attribute transformation (as discretization of numerical attributes and functional transformation). This part can be essential for achieving to goal in entire KDD process and it is usually very project-specific. For instances, in medical examinations, the quotient of attributes might be the most important factor, and not each one by itself. In marketing, we may need to consider effects beyond our control as well as efforts and temporal issues (such as studying the effect of advertising accumulation). However, even if we do not use the right transformation at the beginning, we may obtain a surprising effect that hints to us about the transformation needed (in the next iteration). Thus the KDD process reflects upon itself and leads to an understanding of the transformation needed[17].

Data mining

Data mining contains six popular classes of tasks:

✓Anomaly detection (Outlier/change/deviation detection) – The realizing and recognition of unusual data records

✓Association rule learning (Dependency modeling) – Seeks for relationships between variables. For instance a store may gather data on customer purchasing habits. Using association rule learning, the store can figure out which products and items are frequently bought in a shopping cart and use this information for some purposes like arranging and putting the products which are bought beside each other or for introducing a new product they can put it in a place which is normally viewed by customers.

✓Clustering – Discovering groups and structures in the data that are in some way or another "similar", without using known structures in the data.

✓Classification – Generalizing known structure to use to new data. For instance, an e-mail program might attempt to classify an e-mail as "legitimate" or as "spam".

✓Regression – Attempts to extract a function that is able to model the data with the least possible error.

✓Summarization – Providing a more compact representation of the data set, involve visualization and report generation.

Results interpretation and validation

Data mining can accidentally be misused, and can then give us results that seems to be significant; but which do not really predict future behavior and cannot be reproduced on a new sample of data and possess little use. Often this results from investigating too many hypotheses and not performing proper statistical hypothesis testing. A simple version of this problem in machine learning is known as over fitting, but the same problem can arise at different phases of the process and thus a train/test split - when applicable at all - may not be sufficient to prevent this from happening.

The last step of knowledge discovery from data is to check that the patterns generated by the data mining algorithms are really working in the data set, it means is there exist real pattern even in other datasets.

All of patterns discovered by the data mining algorithms are not true in general. Often happens for data mining algorithms to discover patterns in the training set that are not valid for the general data. This is called over fitting. To avoid this problem, in the evaluation and assessing step, patterns are defined by algorithms applied on test sets which are not used for training and the resulting output is compared to the desired output. For instance, a data mining algorithm tries to specify "spam" from "legitimate" emails would be trained on a training set of sample e-mails. After training phase, the achieved patterns would be applied to the test set of e-mails on which were not in train set. The accuracy of the patterns can then be measured from truly recognized emails.

A bunch of statistical techniques could be used to evaluate the algorithm, such as ROC curves.

If the learned patterns do not meet the desired standards, afterward it is necessary to review all of steps that have done in the procedure of KDD such as changing the pre-processing and data mining steps. But in case that if the learned patterns do meet the desired standards and results are be acceptable, then the final step is to interpret and explain the learned patterns and make them as knowledge which will be usable for next times[17].

So, if we want to divide our mission to 3 main part, first step of our procedure will be pre-processing data. Our data is a data set of around 6000 instances that is prepaid by the German university involve a CSV file which is containing the following columns: User id, mission id, query id, query, normalized query, UTC TIMESTAMP, UNIX TIME of submission, Number of CLICKS per each query, CLICKED URLS for each query. Data mining task wich we will use in our project is classification and our task in pre-processing step is mostly concentrating on defining the features which are necessery for the data mining task.

And, final part of our procedure will be evaluation and interpretation of our model. For the step 2 and 3 (data mining task an evaluation) we will use a software called "Weka" which is developed by Wikato university of Newziland for data ming and machine learning purposes. A list of most well known and famous algorithms for data ming task are collected in software and especially for classification we will have pretty much all useful algorithms. For evaluation of model, weka offers different options and measure which will be enough for our evaluations.

1.4 Weka

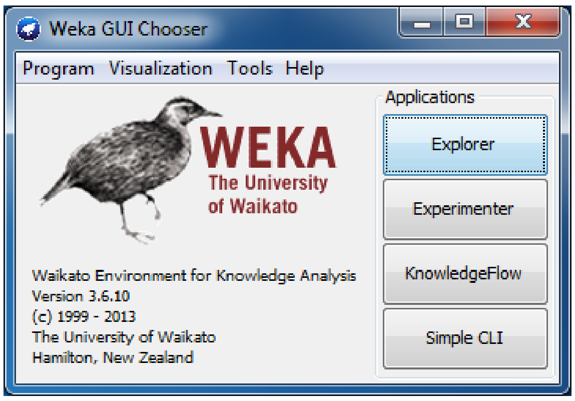
As we mentioned before for the next 2 steps we will work with ”Weka” data mining and machine learning software. It is developed by “Wikato university of New Zealand” and is quite rich of different algorithms for all kind of data mining tasks like classification, clustering, etc.

1.4.1 History of the Weka Project

The WEKA project was funded by the New Zealand government from 1993 up until recently. The original project’s goals as:

“The programme aims to build a state-of-the-art facility for developing techniques of machine learning and investigating their application in key areas of the New Zealand economy. Specifically we will create a workbench for machine learning, determine the factors that contribute towards its successful application in the agricultural industries, and develop new methods of machine learning and ways of assessing their effectiveness.”

The project at first beginning years focused on the developing of interface and instructor of the workbench. Majority of implementations were done in C, with some evaluation written in Prolog programming language. During this period the WEKA1 acronym was made and Attribute Relation File Format (ARFF) was created. The first released version of WEKA was occurred in 1994. May 1998 was the year that the final release of the TCL/TK-based system (WEKA 2.3) and, at the middle of 1999, the 100% Java WEKA 3.0 was released. This non-graphical version of WEKA accompanied the first edition of the data mining book by Witten and Frank. In November 2003, a stable version of WEKA (3.4) was released in anticipation of the publication of the second edition of the book. In the time between 3.0 and 3.4, the three main graphical user interfaces were developed[10]. We do not want to go so much in detail of the Weka software and for more detailed information you can refer to the reference and help of the software[21].



Waikato Environment for Knowledge Analysis
Version 3.6.10
(c) 1999 - 2013
The University of Waikato
Hamilton, New Zealand

Figure 1.2: Weka software interface

Chapter 2

State of The Art

2.1 Query Log Mining

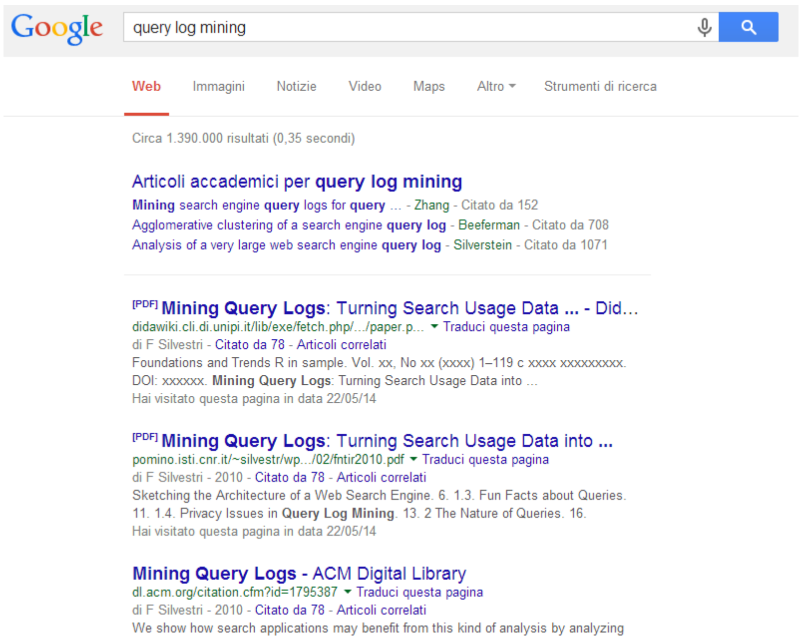
In order to assessing the output and result of a search we can have a look at how users communicate and interact with the Internet by the search engine. For example, it seems interesting to extract what is the pattern that they are following for a search or what they modify their queries according to the results that they get for each query. Queries themselves are not enough to discover and extract user's intent. In addition, one of the most important targets of a search engine is to assessing and evaluating the quality of their suggested links for each query. There are some measure which show some general information about them like the click through rate(the number of clicks a query attract), time-on-page(the time spent on the result page) and scrolling behavior and all of them say that search engines can evaluate theirs results quality according to them.

2.1.1 Query log exploitation

So, now let see if search engines analyze and evaluate these gathered query logs what kind of improvements and services can be obtained by them. Actually, we want to know what the point of mining the query logs is. Here we can define some instances of services that can be offered after these evaluations for a user which have done a query.

Automatic query suggestion

Most of the previous works on the exploitation of query logs deals with the computation of inter-query similarities provide query expansion, suggestion or reformulation.



Circa 1.390.000 risultati (0,35 secondi)

Articoli accademici per **query log mining**
Mining search engine **query** logs for **query** ... - Zhang - Citato da 152
Agglomerative clustering of a search engine **query log** - Beeferman - Citato da 708
Analysis of a very large web search engine **query log** - Silverstein - Citato da 1071

[PDF] **Mining Query Logs**: Turning Search Usage Data ... - Did...
didawiki.cli.di.unipi.it/lib/exe/fetch.php/.../paper.p... Traduci questa pagina
di F Silvestri - Citato da 78 - Articoli correlati
Foundations and Trends R in sample. Vol. xx, No xx (xxxx) 1–119 c xxxx xxxxxxxxx.
DOI: xxxxxx. **Mining Query Logs**: Turning Search Usage Data into ...
Hai visitato questa pagina in data 22/05/14

[PDF] **Mining Query Logs**: Turning Search Usage Data into ...
pomino.isti.cnr.it/~silvestr/wp.../02/fntir2010.pdf Traduci questa pagina
di F Silvestri - 2010 - Citato da 78 - Articoli correlati
Sketching the Architecture of a Web Search Engine. 6. 1.3. Fun Facts about Queries.
11. 1.4. Privacy Issues in **Query Log Mining**. 13. 2 The Nature of Queries. 16.
Hai visitato questa pagina in data 22/05/14

Mining Query Logs - ACM Digital Library
dl.acm.org/citation.cfm?id=1795387 Traduci questa pagina
di F Silvestri - 2010 - Citato da 78 - Articoli correlati
We show how search applications may benefit from this kind of analysis by analyzing

Figure 2.1: A query submitted in a search engine

For example, Beeferman and Breger defined a way to provide suggestion by clustering queries based on the co-occurrence of URLs within the click-through data[1].

Chien and Immorlica proposed a technique to compute inter-query similarity only according to temporal clues[4].

Based on aforementioned techniques, the idea is quite simple: two query are related if they tend to co-occurrence at the same time; so, the similarity of two query is derived from the correlation coefficient of their frequency functions[7].

Re-ranking of search results

Joachims was the first person who used click-through data as re relevance measure about the retrieved results for each query. He used data to define a function for learning of a specialized meta-search engine which later outperformed a commercial search engine[13].

Later bunch of other works were done based on Joachims works and some other ideas.

Other uses for query logs

Query logs also have been used in other things. For example, Chuang and Chien proposed a method to categorize queries submitted to a search engine to assist in the process of building Web Taxonomies[5].

Cucerzan and Brill proposed to use query logs to perform spelling correction[6].

2.2 Search Session Definition

Till now, no precise definition has been defined neither for session nor query session. Actually, seems to be a general agreement about them in the literature. When users interact with a search engine for reaching to their needs, they produce a sequence of queries that are able of being recorded and subsequently analyzed. Therefore, any definition of session should take account of this iterative and evolving, in addition to the underlying existence of user goals[7].

Typical statistical analysis which are done on query logs simply assist the query set for measuring query popularity, term popularity, average query length, distance between repetition of queries or terms, etc. More detailed and depth analysis is search session which is temporal sequences of queries submitted by users[16].

Actually, the first analysis which has done for first time in- depth for assessing search engine queries was conducted by Jansen et al. They did not provide any session definition but the grouped together all queries done by each user[12].

For first time that that a clear definition for a search session provided is possibly that of Silverstein et al[20].

"A session is a series of queries by a single user made within a small range of time; a session is meant to capture a single user's attempt to fill a single information need."

For first time that a clear definition for a search session provided is possibly that of Silverstein et al. A session is a series of queries by a single user made within a small range of time; a session is meant to capture a single user's attempt to fill a single information need. Majority of definitions consider that a session corresponds, at most, to the interval of time extending the first to the last recorded query submitted to a search engine by a certain user in a given day. Moreover, it sounds generally accepted that a period like that does not always corresponds to just one query but to several ones and many of them have relation. Then, from this point to forward the term searching episode and searching session will be used. First one shows the actions are done by a particular user within a search engine during at most, one day. An episode like this can comprise one or more sessions where each of them involves one or more consecutive queries related to on single information need or target.

Therefore, a session from a search engine perspective can be:

1. The whole sequence of queries issued by one user during one single day;
2. The sequence of queries issued by one user since s/he starts the browser until s/he quits;

3. A sequence of queries with no more than a few minutes of inactivity between them. The following literature review takes account of these and other views of search session.

2.3 Analyzing behavior of user in a search session

A series of queries can be a portion of a unique information searching procedure. There are some researches on the effect of set of requests on the search engine side. The most important purpose of this type of analysis is clarifying how users interact with the search engines and how is their next action according to the results showed provided by search engine. One of the interesting thing is to see how users interact with the search engine according to the page request. Users usually just check the results of just the first page and they do not care about the results of the next pages. During the search session a user usually try to rewrite (or modify) queries for achieving the better results. This type of behavior is studied by Lau and Horvitz by categorizing queries according to the seven categories[15].

1. **New**: A query for a topic which is not searched by the user inside of a scope of dataset (one day);
2. **Generalization**: A query on the same topic as the previous one, but searching for more general information than the previous one.
3. **Specialization**: A query on the same topic as the previous one, but searching for more specific information than the previous one.
4. **Reformulation**: A query on the same topic that can be viewed as neither a generalization nor a specialization, but a reformulation of the prior query.
5. **Interruption**: A query on a topic searched on earlier by a user that has been interrupted by a search on another topic.
6. **Request for additional results**: A request for another set of results on the same query from the search service.
7. **Blank** queries: Log entries containing no query.

2.4 Query Classification and Session detection methods

Search engines sometimes use different ways to store queries. Some of them only take one cookie to store an alphanumeric user ID. Some of them also store session ID. Since mid of 2008, Google, AltaVista and Baidu just use user ID, but other hand Yahoo, Ask, Live take use both user and session ID.

In this section we will have a look at different ways and techniques to partition query logs into search sessions, which are short sequences of continuous queries that have dependent single goal for retrieving needed information[7]. There will be two kind of signal can be exploited lonely or in combined way for detecting session boundaries:

1. the time gap between queries
2. the query reformulation patterns

2.4.1 Temporal clues for session boundary detection

Users tend to submit bursts of queries for short periods of time and enter afterwards relatively long periods of inactivity. Therefore, to find out session boundaries, Silverstein et al[20]. suggested applying temporal thresholds. They used a 5 min cutoff: if two queries were less than 5 min apart they would belong to the same session and otherwise to different sessions. This technique is very common because of its simplicity and has been used with different thresholds: 5 min, between 10 and 15 min, and 30 min.

2.4.2 Lexical clues for session boundary detection

Other techniques have suggested about using the content of the queries to figuring out if there is a topic change or not. For this issue different classifications of pattern seeking have been proposed. For the purpose of session boundary detection the patterns that are mentioned above as behavior of users could be interested.

2.4.3 Machine-learning methods to combine temporal and lexical clues

Using search pattern to find out session boundaries will show two main approaches:

1. New search pattern always mention a session border

2. Statistical information can be collected from the query logs to figuring out the probability that this search pattern in reality point a change of session depending on the time gap between the two consecutive queries.

The nature of a New Search pattern mentions that so many topically dependent queries are fragmented into different sessions. Away from the combination to temporal and lexical data the approach has two key sight. First, it needs training data which is judged by human analysis. The information latterly is used to compute the conditional probability of shift given the time interval and search patters.

Second, for combining those probabilities to the Dempster's rule two confidence weights are required; both the weights and the aforementioned threshold are to be obtained by means of genetic algorithms[7, 11].

2.5 Search mission detection

Another point of view is provided by the analysis of Multitasking and Task Switching in query session. Multitasking sessions are the ones that users searching information for different topics at the same time. Some research has done for pointing that how the users have tend to maintain multi-tasking queries.

For example, Ozmutlu et al.[19] show that in 11.4 % of the searches users follow multitasking sessions. Actually, task and multitasking t=is the point that will open another issue in query log analysis and it is Search mission.

Query session is taking care of identifying the series of successive queries that a user does for searching a same information. Nevertheless, assessing just only successive queries will lose some important connections: typical patterns involve interleaving session resulting from a multitasking search behavior as well as hierarchies of different search goals and so-called "missions". For instance a user may shortly interrupt a long search session for weather forecasting, or may check subordinate search targets spanning some days that eventually form a larger mission. So, here again two issue of search session detection and multitasking/mission identification is come up again. The recent approach for session detection is the cascading method[9]. The new cascading method with a new step that uses a Linked Open Data analysis for session detection[8].

2.6 Singletone mission

So far we said that we can go one step forward from the query session to query mission discovery. Imagine we have a session involve bunch of queries. According to the definition of mission inside of a session we will have one or more mission that each of them are for a needed information. But, among them there are queries and missions that even have done in a session with other queries but actually does not have any relation with other queries and mission that are inside the session. In our research we call the singleton queries.

These singleton queries can affect all of tools which are done for query log analysis. Now, actually this is our motivation for doing this thesis and we are sure that finding these singleton missions and being able to extracting them out of session could be incredibly useful for all tools that are working with queries.

Therefore, we are going to get help from Data mining and machine learning methods to apply our idea and use classification task to make a model for discovering these singleton mission according to statistical data that we have about each query.

Chapter 3

Problem Statement

3.1 Background and Notation

Generally speaking, a classification problem can be described as follows.

Let $\mathcal{X}$ be a population of objects, where each object $\mathbf{x_i} \in \mathcal{X}$ is represented by an n-dimensional vector of *features*, i.e., $\mathbf{x_i} = (x_{i,1}, \ldots, x_{i,n})$, and with each feature $x_{i,j}$ describing a particular aspect of the object. In addition, let each $\mathbf{x_i}$ belong to one *class*, labeled as y_i, out of a set of k classes[1], i.e., $y_i \in \mathcal{Y} = \{0, \ldots, k-1\}$.
We call $(\mathbf{x_i}, y_i)$ a *labeled example*, namely any object instance with its associated class label. Suppose $\mathcal{D} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m)\}$ is a set of m labeled examples drawn from the population of objects above, and call it the *gold set*. Also, assume we randomly partition $\mathcal{D}$ in two subsets, i.e., a *training set* ($\mathcal{D}_{\text{train}}$) and a *test set* ($\mathcal{D}_{\text{test}}$), so that:

$$\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}.$$

A *classifier* is a learning algorithm that uses the (labeled) training set to predict which class label $y' \in \mathcal{Y}$ should be assigned to *unseen* instances of the population $\mathbf{x}' \notin \mathcal{D}_{\text{train}}$, i.e., instances which lay outside the training set and whose actual class is unknown.

Specifically, assuming the existence of an unknown target function $f : \mathcal{X} \longmapsto \mathcal{Y}$, which maps each object to its correct class, the goal of a classifier is to exploit the training set $\mathcal{D}_{\text{train}}$ to *learn* another function $h^{\star} : \mathcal{X} \longmapsto \mathcal{Y}$, called *hypothesis* and selected from the space of all possible hypotheses $\mathcal{H}$, which best approximates f.

Since the real target function f is unknown, $h^{\star}$ may assume any value outside

[1] If $k = 2$, we reduce to a binary classification problem.

$\mathcal{D}_{\text{train}}$, namely the space of all possible hypotheses $\mathcal{H}$, which $h^\star$ may be selected from is potentially infinite, thereby making learning computationally unfeasible. However, the introduction of the *Probably Approximately Correct* (PAC) learning theory allows relaxing the learning task as to finding $h^\star \approx f$, *with high probability*. Concretely, to estimate how well an hypothesis $h \in \mathcal{H}$ approximates f, we define a *cost function* $\omega : \mathcal{D}_{\text{train}} \times \mathcal{H} \longmapsto \mathbb{R}$, which intuitively measures the penalty h should pay for incorrectly classifying instances in $\mathcal{D}_{\text{train}}$. Finally, the classification algorithm chooses $h^\star$ among the space of all possible hypotheses, so as to minimize the cost function:

$$h^\star = \operatorname*{argmin}_{h \in \mathcal{H}} \omega(\mathcal{D}_{\text{train}}, h).$$

The overall performance of a classifier can be then evaluated on the remaining portion of the original gold set, i.e., the test set $\mathcal{D}_{\text{test}}$.

Typically, for each labeled example $(\mathbf{x_i}, y_i) \in \mathcal{D}_{\text{test}}$ we do the following. First, we compute the chosen function $h^\star(\mathbf{x_i})$. This would result either in a match, i.e., $h^\star(\mathbf{x_i}) = y_i$ or in an error, i.e., $h^\star(\mathbf{x_i}) \neq y_i$. Finally, we can measure several indicators, such as *true positives* (tp), *true negatives* (tn), *false positives* (fp), and *false negatives* (fp), which in turn allow us to compute *precision*, *recall*, and F_1 scores.

In the following, we assume our population of objects to be the queries of a query log. We also assume the query log is partitioned into a set of (long-term) user sessions, which in turn are parititioned into a set of search missions.

3.2 Problem Definition

Let $\mathcal{Q}$ be a query log made of a set of (long-term) user search sessions $\mathcal{S}_i$.

$$\mathcal{Q} = \{\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{Q}|}\} = \bigcup_{i=1}^{|\mathcal{Q}|} \mathcal{S}_i.$$

Each $\mathcal{S}_i$ is in turn composed of a sequence of queries $q_{i,j}$:

$$\mathcal{S}_i = (q_{i,1}, \ldots, q_{i,|\mathcal{S}_i|}) = \bigcup_{j=1}^{|\mathcal{S}_i|} q_{i,j}.$$

It turns out that:

$$\mathcal{Q} = \bigcup_{i=1}^{|\mathcal{Q}|} \bigcup_{j=1}^{|\mathcal{S}_i|} q_{i,j}.$$

In addition, we assume each $\mathcal{S}_i$ is divided into a set of search missions $\mathcal{M}_i$. Each element of $\mathcal{M}_i$ is actually a subset of queries of the original $\mathcal{S}_i$:

$$\mathcal{M}_i = \{M_{i,1}, \ldots, M_{i,|\mathcal{M}_i|}\},$$

$$M_{i,k} \subseteq \mathcal{S}_i \ \forall k \in \{1, \ldots, |\mathcal{M}_i|\}.$$

It turns out that:

$$\mathcal{S}_i = \bigcup_{k=1}^{|\mathcal{M}_i|} M_{i,k}.$$

Therefore, $\mathcal{M}_i$ is a mission-based partitioning of the original search session $\mathcal{S}_i$. We can thus define the set $\mathcal{M}$ of all the search missions in $\mathcal{Q}$ as follows:

$$\mathcal{M} = \bigcup_{i=1}^{|\mathcal{Q}|} \bigcup_{k=1}^{|\mathcal{M}_i|} M_{i,k}.$$

In addition, let $m : \mathcal{Q} \longmapsto \mathcal{M}$ be a bijection which maps each query to exactly one search mission, i.e., to the search mission which q belongs to. Concretely, if $q \in \mathcal{Q}$ is the generic query of the log and $M \in \mathcal{M}$ is a search mission:

$$m(q) = M \text{ iff } q \in M.$$

Furthermore, we define a boolean function $g : \mathcal{Q} \longmapsto \{0, 1\}$, which states whether a query $q \in \mathcal{Q}$ is part of a *singleton* mission:

$$g(q) = \begin{cases} 1 & \text{if } |m(q)| = 1, \\ 0 & \text{otherwise.} \end{cases}$$

We can learn our binary classifier as follows. Each query $q_i \in \mathcal{Q}$ is represented by an n-dimensional *vector of features*, i.e., $\mathbf{q_i} = (x_{i,1}, \ldots, x_{i,n})$, with each $x_{i,j}$ describing a particular aspect of the query q_i. Eventually, we can build our *gold set* of queries belonging to singleton search mission, i.e., search missions composed of only one query, as follows:

$$\mathcal{D} = \{(\mathbf{q_1}, g(q_1)), \ldots, (\mathbf{q}_{|\mathcal{Q}|}, g(q_{|\mathcal{Q}|}))\} = \bigcup_{q \in \mathcal{Q}} (\mathbf{q}, g(q)).$$

3.3 Features Extraction

A crucial aspect in order for any classifier to be effective is to properly choose the set of features to represent each instance (i.e., each query). In the following, we try to explore those clues that might provide a clear evidence for a query to be actually part of a singleton mission.

In order to generate the vector of features $\mathbf{q}$ for each query q, we consider both *contextual* and *non-contextual* signals. More specifically, the former refers to features that are computed with respect to a subset of all the other queries q', which appear in the same session $\mathcal{S}$of q or, even more broadly, in the whole query log $\mathcal{Q}$. On the other hand, we might also look at each query q individually. For instance, we may want to check whether q is an URL.

In the sections below, we will discuss these two aspects separately.

3.3.1 Non-Contextual Features

– `n_clicks`$(q) \in \mathbb{N}$
  This is a discrete feature which counts the total number of click that have done for each query. q.

– `count_terms`$(q) \in \mathbb{N}$
  This is a discrete feature which counts the total number of terms of the query q.

– `is_valid_domain`$(q, T, D) \in \{0, 1\}$
  Let T be a dictionary of top-level web domains: $T = \{t_1, \ldots, t_{|T|}\}$. In addition, let D be a collection of domain names: $D = \{d_1, \ldots, d_{|D|}\}$.
  This is a boolean feature which states whether at least one of the strings obtained by appending each top-level domain to q turns out to be a valid hostname, that is it checks if: $\exists t_i \in T, d_j \in D \mid q \oplus t_i = d_j$, where $\oplus$ represents the append operator between two strings.

> **Note**: Maybe this feature is useful only for single-term queries. However, there might be some cases where two-terms queries refer to a valid web domain as well, e.g., "`ny times`" may be linked to the valid web domain `nytimes.com`. Also, the dictionary of top-level web domains T can be retrieved from here: `http://en.wikipedia.org/wiki/List_of_Internet_top-level_domains` whereas a list of the most common (US) web domains is available from here: `http://www.alexa.com/topsites/countries/US`.

– `is_wikipedia_anchor`$(q, W) \in \{0, 1\}$
  This is a boolean feature which states if the query q actually links to a Wikipedia article. To this end, we might check if the string representing q appears as an anchor text for one (or more) Wikipedia article.

3.3.2 Contextual (Within-Session) Features

Assuming the query q belongs to the long term session $\mathcal{S} \in \mathcal{Q}$, the following contextual features for q may be computed on the basis of a subset of all the remaining queries $q' \in \mathcal{S}$ or, even more broadly, $q' \in \mathcal{Q}$, such that $q' \neq q$.
Also, let $t(q)$ be the issuing timestamp of each query $q \in \mathcal{Q}$. In addition, let $\prec_\tau$ be a *total order* representing the time-ordering relationship between the queries in $\mathcal{S}$. We thus define $\tau(q, q')$ as the elapsed time between q and q':

$$\tau(q, q') = |t(q) - t(q')|.$$

Note that here we don't take into account the sign of this measure, i.e., $\tau(q, q') = \tau(q', q)$.

- $\texttt{normalized\_count\_terms\_loc}(q, \mathcal{S}) \in \mathbb{R}$,
  $\texttt{normalized\_count\_terms\_glb}(q, \mathcal{Q}) \in \mathbb{R}$
  This is a continuous feature which can be derived by $\texttt{count\_terms}(q)$ above as follows. Let $\texttt{max\_count\_terms\_loc}(\mathcal{S})$ be the number of terms of the query $q' \in \mathcal{S}$ having the maximum number of keywords in session S $\texttt{max\_count\_terms\_glb}(\mathcal{Q})$ having the maximum number of keywords in query log Q , i.e.,

  $$\texttt{max\_count\_terms\_loc}(\mathcal{S}) = \max_{q' \in \mathcal{S}}(\texttt{count\_terms}(q'))$$

  and

  $$\texttt{max\_count\_terms\_glb}(\mathcal{Q}) = \max_{q' \in \mathcal{Q}}(\texttt{count\_terms}(q'))$$

  Then, we can compute:

  $$\texttt{normalized\_count\_terms\_loc}(q, \mathcal{S}) = \frac{\texttt{count\_terms}(q)}{\texttt{max\_count\_terms\_loc}(\mathcal{S})}$$

  $$\texttt{normalized\_count\_terms\_glb}(q, \mathcal{Q}) = \frac{\texttt{count\_terms}(q)}{\texttt{max\_count\_terms\_glb}(\mathcal{Q})}$$

- $\texttt{z\_count\_terms\_loc}(q, \mathcal{S}) \in \mathbb{R}$, $\texttt{z\_count\_terms\_glb}(q, \mathcal{Q}) \in \mathbb{R}$
  This is another way of normalizing the raw $\texttt{count\_terms}(q)$ feature. Indeed, this is the *standard score* which represents the number of standard deviations an observation is above the *population* mean. It is obtained by subtracting the population mean from an individual raw score, and then dividing such difference by the population standard deviation.

We have to think about what "population" here refers to: either it refers to the session $\mathcal{S}$ where q resides or it overall refers to the whole query log $\mathcal{Q}$. Anyway, let the population mean of $\texttt{count\_terms}(q)$ with respect to $\mathcal{S}(\mathcal{Q})$ be $\mu_{\mathcal{S}}$ ($\mu_{\mathcal{Q}}$), and let the population standard deviation with respect to $\mathcal{S}(\mathcal{Q})$ be $\sigma_{\mathcal{S}}$ ($\sigma_{\mathcal{Q}}$), two features are computed as follows:

$$\texttt{z\_count\_terms\_loc}(q, \mathcal{S}) = \frac{\texttt{count\_terms}(q) - \mu_{\mathcal{S}}}{\sigma_{\mathcal{S}}}$$

$$\texttt{z\_count\_terms\_glb}(q, \mathcal{Q}) = \frac{\texttt{count\_terms}(q) - \mu_{\mathcal{Q}}}{\sigma_{\mathcal{Q}}}$$

– $\texttt{session\_length}(q, \mathcal{S}) \subseteq \mathbb{N}$
  This is a discrete feature which measures the total number of queries of the session $\mathcal{S}$. It is straightforward to realize that if $\texttt{session\_length}(q, \mathcal{S}) = 1$ this means that q is the *only* query of the session, and therefore it is surely a singleton. Due to this, maybe we should focus only on those sessions $\mathcal{S} \in \mathcal{Q}$ such that $|\mathcal{S}| > 1$.

– $\texttt{normalized\_session\_length}(q, \mathcal{S}, \mathcal{Q}) \subseteq \mathbb{R}$
  This is a continuous feature which measures the total number of queries of the session $\mathcal{S}$ yet normalized with respect to all the other sessions $\mathcal{S}' \in \mathcal{Q}$.
  Let $\texttt{max\_session\_length}(\mathcal{Q})$ be the length of the session $\mathcal{S}' \in \mathcal{Q}$ having the maximum number of queries, i.e.,

$$\texttt{max\_session\_length}(\mathcal{Q}) = \max_{\mathcal{S}' \in \mathcal{Q}}(\texttt{session\_length}(q, \mathcal{S}'))$$

  Then, we can compute:

$$\texttt{normalized\_session\_length}(q, \mathcal{S}, \mathcal{Q}) = \frac{\texttt{session\_length}(q, \mathcal{S})}{\texttt{max\_session\_length}(\mathcal{Q})}$$

– $\texttt{z\_session\_length}(q, \mathcal{S}, \mathcal{Q}) \subseteq \mathbb{R}$
  Similarly to the $\texttt{z\_count\_terms}(q, \mathcal{Q})$, this feature compute the standard score of the session length. Specifically, if we denote by μ_Q and σ_Q the population mean and standard deviation of the session length, we can compute this feature as follows:

$$\texttt{z\_session\_length} = \frac{\texttt{session\_length}(q, \mathcal{S}) - \mu_{\mathcal{Q}}}{\sigma_{\mathcal{Q}}}$$

– $\texttt{is\_first\_query}(q, \mathcal{S}) \in \{0, 1\}$

This is a boolean feature which states whether q appears as the very *first* query of the session $\mathcal{S}$.

- $\texttt{is\_last\_query}(q, \mathcal{S}) \in \{0, 1\}$
  This is a boolean feature which states whether q appears as the very *last* query of the session $\mathcal{S}$.

- $\texttt{time\_elapsed\_from\_prev}(q_i, \mathcal{S}) \in \mathbb{N}$
  Let q_i be the i-th query of $\mathcal{S}$ and let $\perp$ be a default value, which may either be 0 or ∞. This feature equals to:

$$\begin{cases} \perp & \text{if } i = 1, \\ \tau(q_i, q_{i-1}) & \text{otherwise.} \end{cases}$$

- $\texttt{time\_elapsed\_to\_next}(q_i, \mathcal{S}) \in \mathbb{N}$
  Let q_i be the i-th query of $\mathcal{S}$ and let $\perp$ be a default value, which may either be 0 or ∞. This feature equals to:

$$\begin{cases} \perp & \text{if } i = |\mathcal{S}|, \\ \tau(q_i, q_{i+1}) & \text{otherwise.} \end{cases}$$

- $\texttt{z\_time\_elapsed\_from\_prev\_loc}(q_i, \mathcal{S}) \in \mathbb{R}$,
  $\texttt{z\_time\_elapsed\_from\_prev\_glb}(q_i, \mathcal{Q}) \in \mathbb{R}$
  This is a way of normalizing the raw $\texttt{time\_elapsed\_from\_prev}(q_i, \mathcal{S})$ feature on the basis of population mean and standard deviation. Again, we have to think about what "population" here refers to: either it refers to the session $\mathcal{S}$ where q_i resides or it overall refers to the whole query log $\mathcal{Q}$.
  Anyway, let the population mean of $\texttt{time\_elapsed\_from\_prev}(q_i, \mathcal{S})$ with respect to $\mathcal{S}(\mathcal{Q})$ be $\mu_{\mathcal{S}}$ ($\mu_{\mathcal{Q}}$), and let the population standard deviation with respect to $\mathcal{S}(\mathcal{Q})$ be $\sigma_{\mathcal{S}}$ ($\sigma_{\mathcal{Q}}$), two features are computed as follows:

$$\texttt{z\_time\_elapsed\_from\_prev\_loc}(q_i, \mathcal{S}) = \frac{\texttt{time\_elapsed\_from\_prev}(q_i, \mathcal{S}) - \mu_{\mathcal{S}}}{\sigma_{\mathcal{S}}}$$

$$\texttt{z\_time\_elapsed\_from\_prev\_glb}(q_i, \mathcal{Q}) = \frac{\texttt{time\_elapsed\_from\_prev}(q_i, \mathcal{Q}) - \mu_{\mathcal{Q}}}{\sigma_{\mathcal{Q}}}$$

- $\texttt{z\_time\_elapsed\_to\_next\_loc}(q_i, \mathcal{S}) \in \mathbb{R}$,
  $\texttt{z\_time\_elapsed\_to\_next\_glb}(q_i, \mathcal{Q}) \in \mathbb{R}$

Similarly to the scores above we can compute the following features:

$$\texttt{z\_time\_elapsed\_to\_next\_loc}(q_i, \mathcal{S}) = \frac{\texttt{time\_elapsed\_to\_next}(q_i, \mathcal{S}) - \mu_{\mathcal{S}}}{\sigma_{\mathcal{S}}}$$

$$\texttt{z\_time\_elapsed\_to\_next\_glb}(q_i, \mathcal{Q}) = \frac{\texttt{time\_elapsed\_to\_next}(q_i, \mathcal{Q}) - \mu_{\mathcal{Q}}}{\sigma_{\mathcal{Q}}}$$

– n_of_equal_queries$(q_i, \mathcal{S}) \in \mathbb{N}$
This feature states $q_i \in \mathcal{S}$ how many equal query has in session§. Note that this feature is defined only queries that belong to the same user session $\mathcal{S}$.

– z_n_of_equal_queries_loc$(q_i, \mathcal{S}) \in \mathbb{R}$
This a way for normalizing the equal number of queries of each query in a session.

– z_n_of_equal_queries_glb$(q_i, \mathcal{S}) \in \mathbb{R}$
This a way for normalizing the equal number of queries of each query in a query log $\mathcal{Q}$.

3.3.3 Semantic Contextual (Within-Session) Features

Till now we worked on features which are all non semantic. In this part we are going to define some semantic features that for each query should measured according to the others queries which are in the same session or in query log.

– min_wiki_similarity$(q_i, \mathcal{S}) \in \mathbb{R}$, max_wiki_similarity$(q_i, \mathcal{S}) \in \mathbb{R}$
This function returns the cosine similarity between the "wikified" expansions of both input strings It assumes the two strings are already represented in the Wikipedia vector space.

– min_haming_distance$(q_i, \mathcal{S}) \in \mathbb{R}$, max_haming_distance$(q_i, \mathcal{S}) \in \mathbb{R}$
The Hamming distance between two strings computes the number of positions where a character mismatch occurs For binary (i.e., 0/1) strings this is equal to the number of 1s in s XOR b The function takes as input two strings and an optional parameter which specifies whether the final score has to be normalized with respect to the length of the string (default = False) NOTE: the Hamming distance is defined ONLY when the two strings have the same length!

– min_haming_similarity$(q_i, \mathcal{S}) \in \mathbb{R}$, max_haming_similarity$(q_i, \mathcal{S}) \in \mathbb{R}$
This is just the complimentary score of the Hamming distance above.

– `min_jacard_char_level_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$,
`max_jacard_char_level_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$
This is the Jaccard coefficient which is always in [0,1] range. It is computed as follows:

$$J = |set(a) AND set(b)|/|set(a) OR set(b)|$$

– `min_jacard_word_level_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$,
`max_jacard_word_level_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$
This is the Jaccard coefficient which is always in [0,1] range. It is computed as follows:

$$J = |set(a) AND set(b)|/|set(a) OR set(b)|$$

– `min_sorensen_dice_char_level_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$,
`max_sorensen_dice_char_level_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$
This is the Sorensen-Dice coefficient which is always in [0,1] range. It is computed as follows:

$$QS = 2C/(|s| + |t|) where C = |set(a) AND set(b)|$$

– `min_sorensen_dice_word_level_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$,
`max_sorensen_dice_word_level_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$
This is the Sorensen-Dice coefficient which is always in $[0, 1]$ range. It is computed as follows:

$$QS = 2C/(|s| + |t|) where C = |set(a) AND set(b)|$$

– `min_longest_common_substring`$(q_i, \mathcal{S}) \in \mathbb{R}$,
`max_longest_common_substring`$(q_i, \mathcal{S}) \in \mathbb{R}$
This is the length of the longest common substring between two strings s and t.

– `min_longest_common_subsequence`$(q_i, \mathcal{S}) \in \mathbb{R}$,
`max_longest_common_subsequence`$(q_i, \mathcal{S}) \in \mathbb{R}$
This is the length of the longest common subsequence between two strings s and t Differently from the longest common substring above, the sequence of common chars here don't need to be one next to the other.

- `min_levenshtine_distance`$(q_i, \mathcal{S}) \in \mathbb{R}$,
  `max_levenshtine_distance`$(q_i, \mathcal{S}) \in \mathbb{R}$
  The Levenshtein distance between two strings computes the minimum number of character-edit operations (i.e., deletion, insertion, and substitution)that are required to change one string to the other The function takes as input two strings and an optional parameter which specifies whether the final score has to be normalized with respect to the length of the longer string (default = False).

- `min_levenshtein_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$,
  `max_levenshtein_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$
  This is just he complimentary score of the Levenshtein distance above Note that if not normalized this returns the difference between the number of characters of the longer string and the levenshtein distance above Intuitively, the levenshtein distance between two strings (s,t) would require at most a number of character-edit operations which is equal to the length of the longest string between s and t.

- `min_jaro_winkler_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$,
  `max_jaro_winkler_similarity`$(q_i, \mathcal{S}) \in \mathbb{R}$
  and

- `min_jaro_winkler_distance`$(q_i, \mathcal{S}) \in \mathbb{R}$,
  `max_jaro_winkler_distance`$(q_i, \mathcal{S}) \in \mathbb{R}$
  The Jaro-Winkler similarity is a score in $[0, 1]$ which uses the Jaro distance as its building block It counts the matching characters of the two strings as well as the transpositions needed Matches are counted by looping over the shortest string. For each character of the shortest string in position i, we look if the same character occurs in the longest string either at the same position (i) or in a window of characters before and after i, i.e., $window = int(math.floor(max(|s|, |t|)/2.0) - 1)$ Also, matches that occur in a different order between the two strings are considered to be transposed The Jaro-Winkler similarity is computed as:

  $$JW(s,t) = J(s,t) + (prefix\_length * prefix\_scale * (1 - J(s,t)))$$

  where $J(s,t)$ is the Jaro distance, and $prefix\_length$ and $prefix\_size$ two tuning parameters.

Chapter 4

Modeling and Evaluation of models

At the end of Feature Extraction section we made 48 attributes for each query. From this 48 attributes, 4 of them are contextual features which are evaluating independently from the other queries and just are measures for each of them, 18 of them are contextual within session which means the values are not independent for each query and are calculated according to the other queries in a session and finally 26 of them are semantic contextual which are just Minimum of Maximum of values that each query has in compare to other queries in same session.

4.1 Data mining task (Classification)

Now we have done pre-processing step and we can switch to the next step of our KDD procedure. As we mentioned before Data mining and Machine Learning have a lot of overlapping techniques and actually now after extracting the features we will use some methods of machine learning to use these features to extract models by classification algorithms that will be our patterns in data set.

4.2 Evaluating learned model

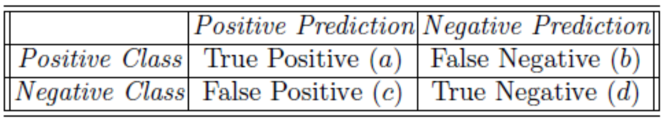
Since we have different types of classification algorithm, we can try all of them which are suitable for numeric data sets like ours to make a model but the point is finding the best one for our problem. Here the importance of model evaluation pups up and plays the main role in our procedure.

During this evaluation procedure a bunch of problems can happen. For example having a limited number of instances makes modeling and testing procedure more difficult. Sometimes the instances are classified imbalance and there is a big difference between the proportions of classes. Therefore, all these issues must take into account during the evaluation progress.

4.3 Commonly-accepted performance evaluation measures

Different evaluation measures assess different characteristics of machine learning and data mining algorithms. The empirical evaluation of algorithms and classifiers is a matter of on-going debate between researchers. Supervised Machine Learning (ML) has several ways of evaluating the performance of learning algorithms and the classifiers they produce. Measures of the quality of classification are built from a confusion matrix which records correctly and incorrectly recognized examples for each class.

Confusion matrix is a good tool for summarizing different types of error. Here we can see a confusion matrix for a two class data set which classes labeled as Positive and Negative which is exactly same as our case.



	Positive Prediction	*Negative Prediction*
Positive Class	True Positive (a)	False Negative (b)
Negative Class	False Positive (c)	True Negative (d)

Figure 4.1: Different types of errors and hits for a two classes problem.

Calculatable measure form confusion matrix are Accuracy and Error rate that can be used for performance evaluation.

$$Errorrate = \frac{fp + fn}{tp + fp + fn + tn}$$

$$accuracy = \frac{tp + tn}{tp + fp + fn + tn} = 1 - Errorrate$$

4.3.1 Why Accuracy and Error Rate are Inadequate Performance Measures for Imbalanced Data Sets

The error rate and the accuracy are measures that more often are been used for calculating the performance of learning systems. However, when the probabilities of the classes are different, such measure may be misleading. For example, it is straightforward to create a classifier having 90 % accuracy (or 10 % error rate) if the distribution of classes be consequently 90 % and 10% of total number of cases, by simply labelling every new case as belonging to the majority class. Another problem about using accuracy (or error rate) is that these measures consider different classification errors as equally important. For example, a sick patients diagnosed as healthy might be a fatal error while a healthy patience diagnosed as sick is considered a much less serious error since this mistake can be corrected in future exams. On domains where misclassification cost is relevant, a cost matrix could be used. A cost matrix defines the misclassification cost, i.e., a penalty for making a mistake for each different type of error. In this case, the purpose of the classifier is to minimize classification cost instead of error rate.

The vast majority of ML research focus on the settings where the examples are assumed to be identically and independently distributed (IID). This is the case we focus on in this study. Classification performance without focusing on a class is the most general way of comparing algorithms. It does not favor any particular application. The introduction of a new learning problem inevitably concentrates on its domain but omits a detailed analysis. Thus, the most used empirical measure, accuracy, does not distinguish between the number of correct labels of different classes:

On contrary, two measures that separately estimate a classifier's performance on different classes are:

$$sensivity = \frac{tp}{tp + fn}$$

and

$$specifity = \frac{tn}{fp + tn}$$

Which Sensitivity mostly refers to the correctly classified of positive class and is often called Recall. And specificity which essentially focuses on negative group.

Sensitivity and specificity are often employed in bio- and medical applications and in studies involved image and visual data. Focus on one class is mostly taken in text classification, information extraction, natural language processing, and bioinformatics. In these areas of application the number of examples belonging to one class is often substantially lower than the overall number of examples. The experimental setting is as follows: within a set of classes there is a class of special interest (usually positive). Other classes are either left as is – multi-class classification – or combined into one – binary classification. The measures of choice calculated on the positive class:

$$precision = \frac{tp}{tp + fp}$$

$$recall = \frac{tp}{tp + fn} = sensivity$$

$$F - measure = \frac{(\beta^2 + 1) * precision * recall}{\beta^2 * precision + recall}$$

All three measures distinguish the correct classification of labels within different classes. They concentrate on one class (positive examples). Recall is a function of its correctly classified examples (true positives) and its misclassified examples (false negatives). Precision is a function of true positives and examples misclassified as positives (false positives). The F-score is evenly balanced when $\beta = 1$. It favors precision when $\beta > 1$, and recall otherwise.

4.3.2 What is a good Performance measure for our data set

A comprehensive evaluation of classifier performance can be obtained by the ROC curves. In signal detection theory, a receiver operating characteristic (ROC), or ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate against the false positive rate at various threshold settings. (The true-positive rate is also known as sensitivity in biomedicine, or recall in machine learning. The false-positive rate is also known as the fall-out and can be calculated as 1 - specificity). The ROC curve is thus the sensitivity as a function of fall-out. In general, if both of the probability distributions for detection and false alarm are known, the ROC curve can be generated by plotting the cumulative distribution function (area under the probability distribution from -inf to +inf) of the detection probability in the y-axis versus the cumulative distribution function of the false-alarm probability in x-axis.

ROC analysis provides tools to select possibly optimal models and to discard suboptimal ones independently from (and prior to specifying) the cost context or the class distribution. ROC analysis is related in a direct and natural way to cost/benefit analysis of diagnostic decision making.

The ROC curve was first developed by electrical engineers and radar engineers during World War II for detecting enemy objects in battlefields and was soon introduced to psychology to account for perceptual detection of stimuli. ROC analysis since then has been used in medicine, radiology, biometrics, and other areas for many decades and is increasingly used in machine learning and data mining research.

The ROC is also known as a relative operating characteristic curve, because it is a comparison of two operating characteristics (TPR and FPR) as the criterion changes. The contingency table can derive several evaluation "metrics" (see infobox). To draw an ROC curve, only the true positive rate (TPR) and false positive rate (FPR) are needed (as functions of some classifier parameter). The TPR defines how many correct positive results occur among all positive samples available during the test. FPR, on the other hand, defines how many incorrect positive results occur among all negative samples available during the test.

A ROC space is defined by FPR and TPR as X and Y axes respectively, which depicts relative trade-offs between true positive (benefits) and false positive (costs). Since TPR is equivalent to sensitivity, and FPR is equal to 1-specifity, the ROC graph is sometimes called the sensivity vs (1-specifity) plot. Each prediction result or instance of a confusion matrix represent one point in the ROC space.

The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). The (0,1) point is also called a perfect classification. A completely random guess would give a point along a diagonal line (the so-called line of no-discrimination) from the left bottom to the top right corners (regardless of the positive and negative base rates). An intuitive example of random guessing is a decision by flipping coins (heads or tails). As the size of the sample increases, a random classifier's ROC point migrates towards (0.5,0.5).

The diagonal divides the ROC space. Points above the diagonal represent good classification results (better than random), points below the line poor results (worse than random). Note that the output of a consistently poor predictor could simply be inverted to obtain a good predictor.

4.3.3 Area under the curve

When using normalized units, the area under the curve (often referred to as simply the AUC, or AUROC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one (assuming 'positive' ranks higher than 'negative'). This can be seen as follows: the area under the curve is given by (the integral boundaries are reversed as large T has a lower value on the x-axis)

$$A = \int_{\infty}^{-\infty} y(T)x'(T)\,dT = \int_{\infty}^{-\infty} \mathrm{TPR}(T)\mathrm{FPR}'(T)\,dT = \int_{-\infty}^{\infty} \mathrm{TPR}(T)P_0(T)\,dT =$$

$\langle \text{TPR} \rangle$. The angular brackets denote average from the distribution of negative samples.

4.4 Training and Testing

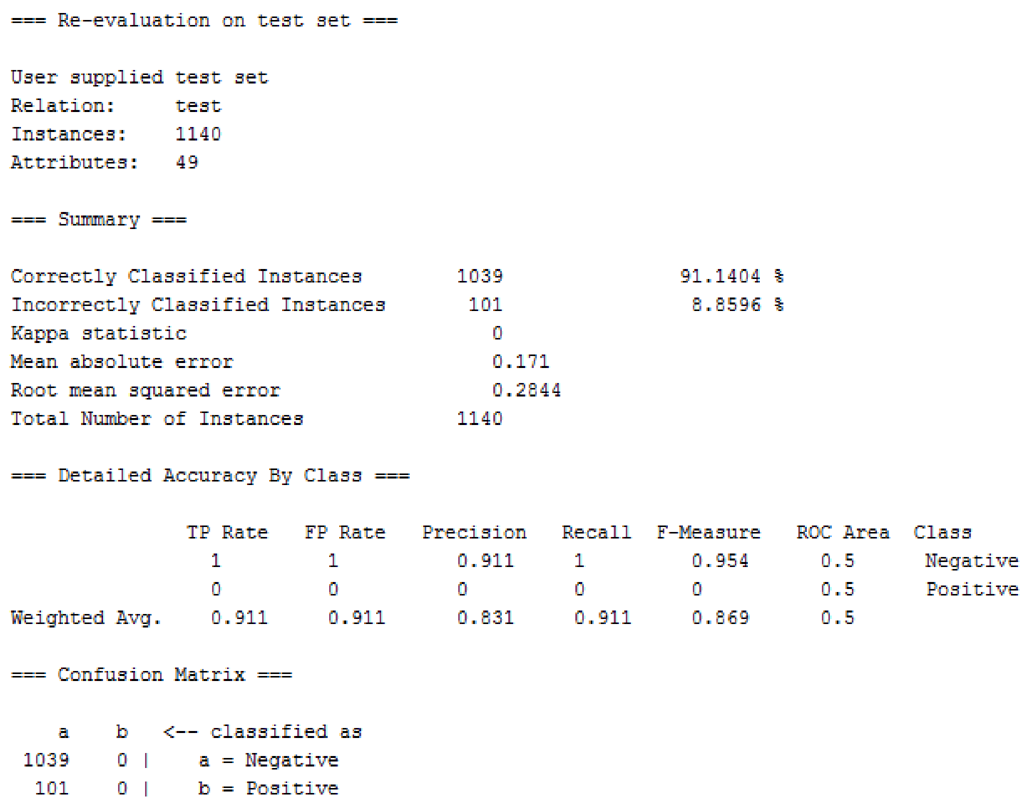
For classification problems, normally performance of model is measured as error rate. So simply, if the model predict the class of each class correctly, it is measured as success and otherwise error. And error rate is calculated by dividing the errors over the whole number of instances. Now the point is this, unfortunately we cannot judge about the performance of a model just according to the error rate of learned model on training set because the classifier has been learned from the same data and performance evaluation based on it will be so optimistic. Because sometime may be we get good result from a model but this good results may not be same on another independent data set. For example because of existence of noise if we make a model with a high accuracy it will lead problem because of over fitting and leaning from noise will mislead our classifier on other real data sets. Hence, for assessing the performance of our learned model we need to evaluate the error rate of the model based on a data which was not a part of training set which is called as test set. Well now if there is large amount of data is available we will not have any problem and we can use a large part as training set and then another independent data as test set. But if we cannot provide good amount of data our modeling will not be good enough. Because large training instance will give a good model. In the cases with limited size of data a certain amount is hold out as training set and the rest will used as test set, this way is called holdout procedure. In our case as we are using the data set provided by the German university form 6383 instance, we have a reasonable size of data. Now point is this, which portion will be suitable for the holdout procedure. As we know for finding a good classifier we should use as much as possible data as training set and for figuring out the good error rate we want to use as much as possible data as test set. Normally, in machine earning and data miming problems of the most reasonable portions is $80 - 20$, which means allocation 80 percent of whole data to training set and the rest 20 percent as test set. Before dividing the data set to training and test set we should shuffle the data. Because after dividing data set, may be the sample for training or testing not be representative for the class distribution. In other word, it is possible that the majority of one class be in training (or test) set and this will absolutely will affect the final model. So, shuffling the data set before dividing it to training and test set can enervate this possibility. Also a more general way to mitigate any bias cause by the particular sample chosen is to repeat the whole process, training and testing, several times with different random samples. The error rate on the different iteration are averaged to yield an overall error rate. An important statistical technique called Cross Validation could be used for this purpose. In cross validation, we decide about a fixed number of folds or parts of data. For example one of the most common division is 10 fold cross validation. Suppose that data is splited to 10 fold or portion. These portion have not to be exactly equal and according to the size of data there could be a tolerance for the size portions. In each turn of these portions is used as test set and the rest will used as training set. In next iteration another portion will be treated as test set and the previous test portion will take into account as training set with other portions. This procedure will have done for 10 time and it is called 10 fold cross validation. Actually training procedure is done for 10 times in this way and the testing model is done on the portion which was out of training set during the learning procedure. At the end 10 error estimates are averaged to yield an overall estimate.

As we said we have a data set with 638 instances, for enervating the problem of equal representation of class in data set, we shuffle the whole data set before dividing it. Then we will use the 80 percent of data set as training set and the 20 percent will be our test set for evaluating the produced predictor. But, our evaluation way will be a mixed of these 2 ways that are mentioned. Here before testing the models by test set we use the other evaluation method which is cross validation for finding the best modeling algorithm for our problem. The candidate modeling algorithms will be used and evaluated by 10 fold cross validation. At this step our assessment about the algorithms will be about error rate and accuracy of each model. As these measure are calculated by averaging the error rates of 10 iteration and 10 times training procedure, for approving the calculated error rates we will do one more time the procedure of learning by using whole training set and testing the output model by the allocated test set which is 20 percent of our initial data set. Actually, learning procedure will be done for the selected algorithms 11 time (10 time during the 10 fold cross validation and last one on whole training set).

Candidate algorithms with high accuracy and error rate As we told before first step of our modeling procedure was assessing most common classification algorithms and modeling our training set by 10 fold cross validation. Among all these common algorithms according to the error rate and accuracy we chose 4 algorithms and went one step more and did the last learning phase which was training the candidate algorithm base on whole training set and final evaluation and test is testing the model on the test set which is the 20 divided percent of whole data set.

4.4.1 ZeroR Algorithm

For knowing what is the base line accuracy of the data set simply by trying the "ZeroR" classifier we can get it. So, the reason why we get it at first step is that actually it could be a simple but a good mood for measuring other classifiers. Actually, since 91 % of our data set is from class "Negative", it is easy to put all instances as Negative and it simply give 91% of accuracy from all Negative classes which all have put in Negative class.



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1039               91.1404 %
Incorrectly Classified Instances       101                8.8596 %
Kappa statistic                          0
Mean absolute error                      0.171
Root mean squared error                  0.2844
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         1         0.911     1         0.954      0.5       Negative
                 0         0         0         0         0          0.5       Positive
Weighted Avg.    0.911     0.911     0.831     0.911     0.869      0.5

=== Confusion Matrix ===

    a    b   <-- classified as
 1039    0 |    a = Negative
  101    0 |    b = Positive
```

Figure 4.2: ZeroR classification Algorithm results

The predictive performance of the model with detail is showed in the right-hand Classifier output frame. The Confusion Matrix for the model is provided at the bottom side of the Classifier output. We can see from it that all instances have been classified as “Negative”. It is clear that such trivial model is useless and it cannot be used for discovering “Positive” instances which are our desired goal. As you see the accuracy of the model (Correctly Classifieds Instances) of this model is very high: 91.14 %. This fact clearly indicates that the accuracy cannot be used for evaluating the usefulness of classification models built using unbalanced datasets like ours which there is a huge difference between majority and minority class.

4.5 Most Influential Data Mining Algorithms

After introducing the evaluation methods we started to modeling by different algorithm and all of them according to the explained evaluation methods we chose 4 algorithm which have better results compare to others. Actually as we mentioned before we trained all of possible classification algorithms and tested them by 10 fold cross validation and these 4 algorithms are chosen among the them and the last step for proving the performance of these algorithms was retraining the selected algorithm by doing one more training by whole training set and testing it by provided test set.

4.5.1 Bayesian network

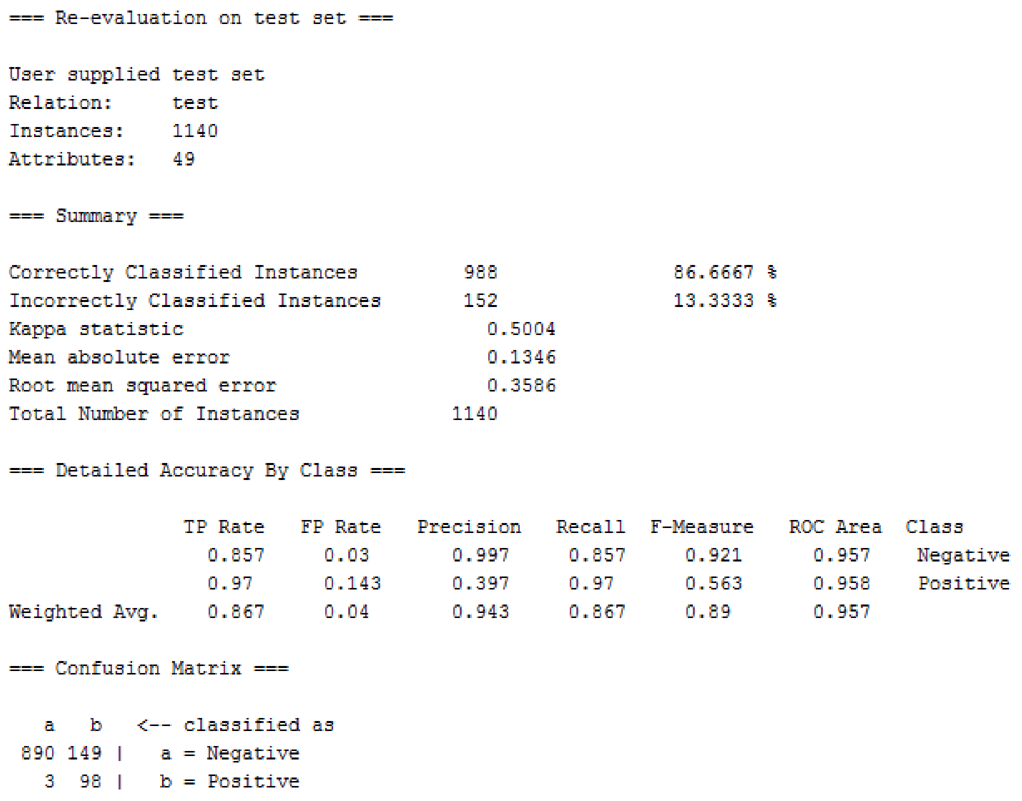
Bayesian networks is a statistical method for Data Mining, a statistical method for discovering valid, novel and potentially useful patterns in data.Bayesian networks are used to represent essential information in databases in a network structure. The network consists of edges and vertices, where the vertices are events and the edges relations between events. Bayesian networks are easy to interpret for humans, and are able to store causal relationships , that is, relations between causes and effects. The networks can be used to represent domain knowledge, and it is possible to control inference and produce explanations on a network.

Let $U = x_1, ..., x_n, n \geq 1$ be a set of variables. A Bayesian network B over a set of variables U is a network structure BS, which is a directed acyclic graph (DAG) over U and a set of probability tables $BP = p(u|pa(u))|u \in U$ where $pa(u)$ is the set of parents of u in BS. A Bayesian network represents a probability distributions $P(U) = Qu \in Up(u|pa(u))$. Inference algorithm to use a Bayesian network as a classifier, one simply calculates $argmaxyP(y|x)$ using the distribution P(U) represented by the Bayesian network. Now note that

$$P(y|x) = P(U)/P(x) \infty P(U) = Yu \in Up(u|pa(u))(1)$$

And since all variables in x are known, we do not need complicated inference algorithms, but just calculate (1) for all class values.

After explaining the Bayesian Network, let try it on our data sets. We do modeling with the prepaid training set with 10 fold cross validation and the summary of the modeling is in below.



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances         988               86.6667 %
Incorrectly Classified Instances       152               13.3333 %
Kappa statistic                          0.5004
Mean absolute error                      0.1346
Root mean squared error                  0.3586
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.857     0.03       0.997     0.857     0.921      0.957    Negative
                 0.97      0.143      0.397     0.97      0.563      0.958    Positive
Weighted Avg.    0.867     0.04       0.943     0.867     0.89       0.957

=== Confusion Matrix ===

   a   b   <-- classified as
 890 149 |   a = Negative
   3  98 |   b = Positive
```

Figure 4.3: Bayesian Network Algorithm

In summary with almost 13.33 % error rate we have a good accuracy. Taking a look at the confusion matrix shows the algorithm had a reasonable output. This algorithm classifies the positive class more successfully with 98 correctly classifies instances over 101 instances and just with 3 incorrectly classified. And it give a high Recall with 0.97 for positive class. For Negative class this algorithm works a bit weaker than positive class and classified 890 over 1039 negative instances and had error for 149 instances. It give 0.85 for recall. But having 149 wrongly classified negative class as positive decrease the precision of positive class for this algorithm which is almost 0.4. Then as we mentioned before for imbalanced data, as there is a big difference between the number of instances for each class may be these evaluation measures could be a bit confusing but the ROC curve measure shows more reasonable the performance of our classifier.

4.5.2 LogitBoost

LogitBoost is a boosting algorithm formulated by Jerome Friedman, Trevor Hastie, and Robert Tibshirani. The original paper[1] casts the AdaBoost algorithm into a statistical framework. Specifically, if one considers AdaBoost as a generalized additive model and then applies the cost functional of logistic regression, one can

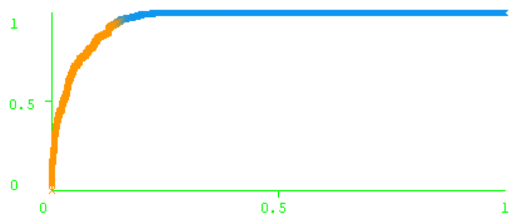


Figure 4.4: bayesian network-threshold-positive

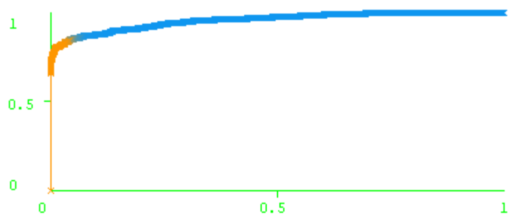


Figure 4.5: bayesian network-threshold-negative

derive the LogitBoost algorithm. LogitBoost can be seen as a convex optimization. Specifically, given that we seek an additive model of the form

$$f = \sum_t \alpha_t h_t$$

the LogitBoost algorithm minimizes the logistic loss:

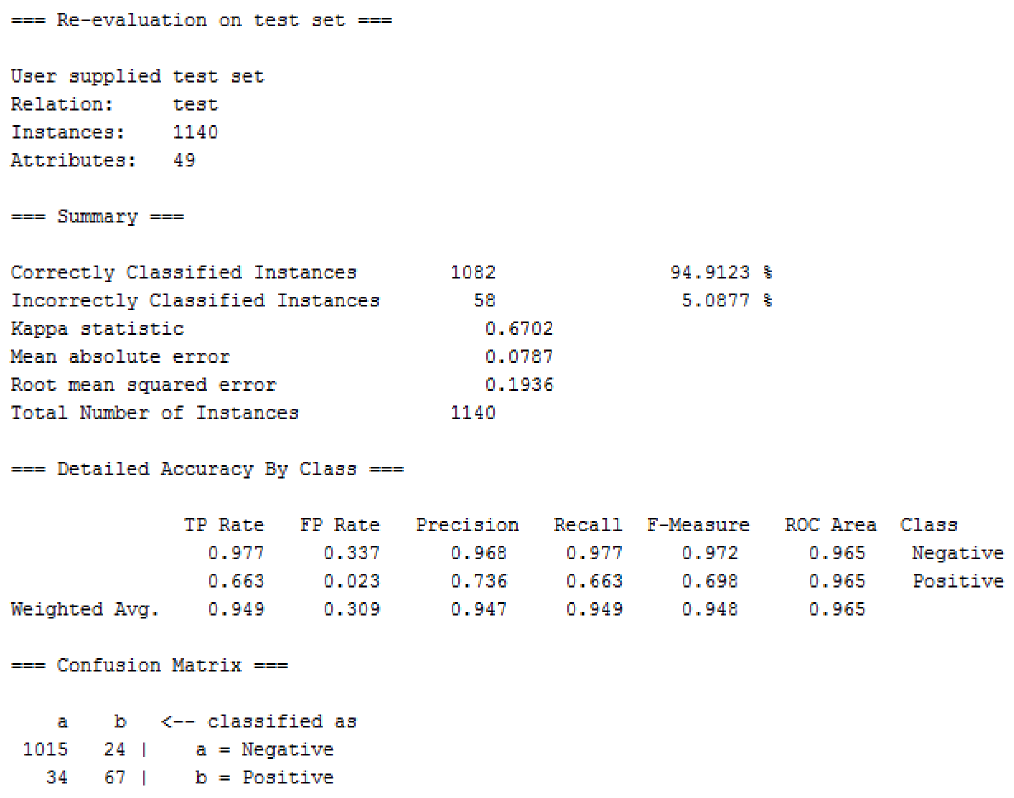
$$\sum_i \log\left(1 + e^{-y_i f(x_i)}\right)$$

We are going to try this algorithm on our data set and see how it works. The summary bellow is shows the classification of our test data by the model which is produced through learning the training set. As we can see the total accuracy is very high and s around 95% and just 5% of error but as we mentioned before we are going to check other measures. According to the confusion matrix there are 24 instances of non single query classified as singleton and 34 singleton query classified as non singleton query. Precision of 0.97 for Negative class is super good but the errors happen for classifying the positive classes make the precision of positive class less and 0.74. Same for recall which is 0.98 for negative class and almost 0.7 for positive.

About ROC area fortunately we have great results and 0.96 for this algorithm is a good result.

4.5.3 Threshold Selector with Logistic Algorithm

A meta-classifier that selecting a mid-point threshold on the probability output by a Classifier. The midpoint threshold is set so that a given performance measure is optimized. Currently this is the F-measure. Performance is measured either on the training data, a hold-out set or using cross-validation. In addition, the probabilities returned by the base learner can have their range expanded so that the output probabilities will reside between 0 and 1 (this is useful if the scheme normally produces probabilities in a very narrow range). By default this algorithm chose the



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1082               94.9123 %
Incorrectly Classified Instances        58                5.0877 %
Kappa statistic                          0.6702
Mean absolute error                      0.0787
Root mean squared error                  0.1936
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.977     0.337      0.968     0.977     0.972      0.965    Negative
                 0.663     0.023      0.736     0.663     0.698      0.965    Positive
Weighted Avg.    0.949     0.309      0.947     0.949     0.948      0.965

=== Confusion Matrix ===

    a    b   <-- classified as
 1015   24 |    a = Negative
   34   67 |    b = Positive
```

Figure 4.6: LogitBosst Algorithm

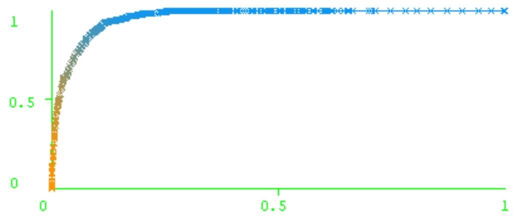


Figure 4.7: logitboost-threshold-positive

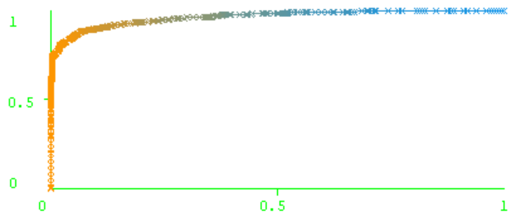
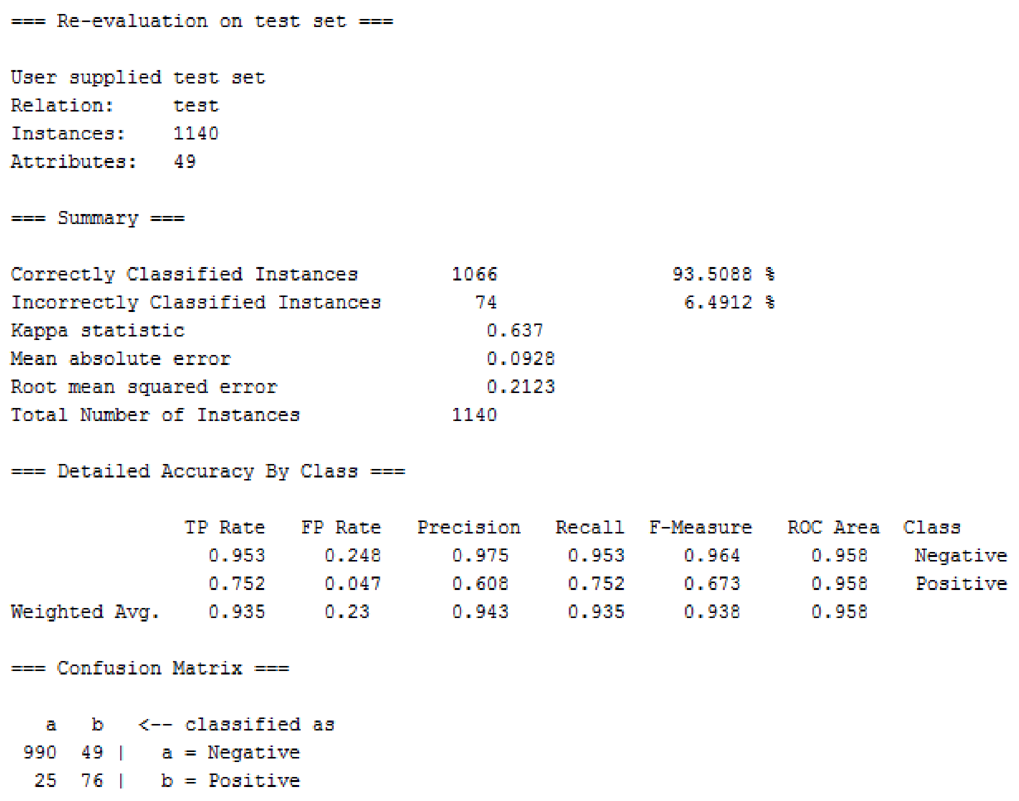


Figure 4.8: logitboost-threshold-negative

Logistic regression classifier and we even we tried other classifiers we found out the Logistic more efficient than others in this algorithm.

Down below is the summary of re-evaluating the test set with the model based on logistic regression classifier. First impression from the accuracy and error rate with 94% and 6% sound a good result. Taking a look at the confusion matrix shows that algorithm with 49 wrongly classified non singleton query as singleton has done good job and worked a bit weakly in classifying the singleton queries with 35 wrongly classified queries. Precision of Negative class is 0.97 which for positive class is 0.6 and recall is 0.95 and 0.75 for negative and positive classes respectively. Next and more important measure for us is ROC area which 0.95 sound really good in this



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1066               93.5088 %
Incorrectly Classified Instances        74                6.4912 %
Kappa statistic                          0.637
Mean absolute error                      0.0928
Root mean squared error                  0.2123
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.953     0.248     0.975     0.953     0.964      0.958     Negative
                 0.752     0.047     0.608     0.752     0.673      0.958     Positive
Weighted Avg.    0.935     0.23      0.943     0.935     0.938      0.958

=== Confusion Matrix ===

   a   b   <-- classified as
 990  49 |   a = Negative
  25  76 |   b = Positive
```

Figure 4.9: Threshold Selector with Logistic algorithm

algorithm.

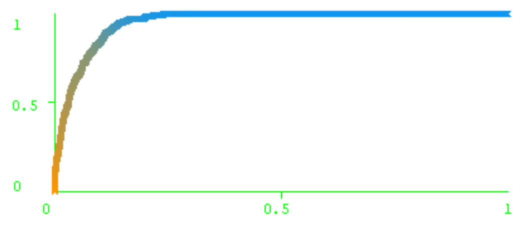


Figure 4.10: Threshold Selector ROC Curve for Positive class

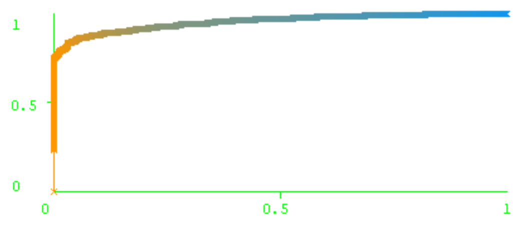
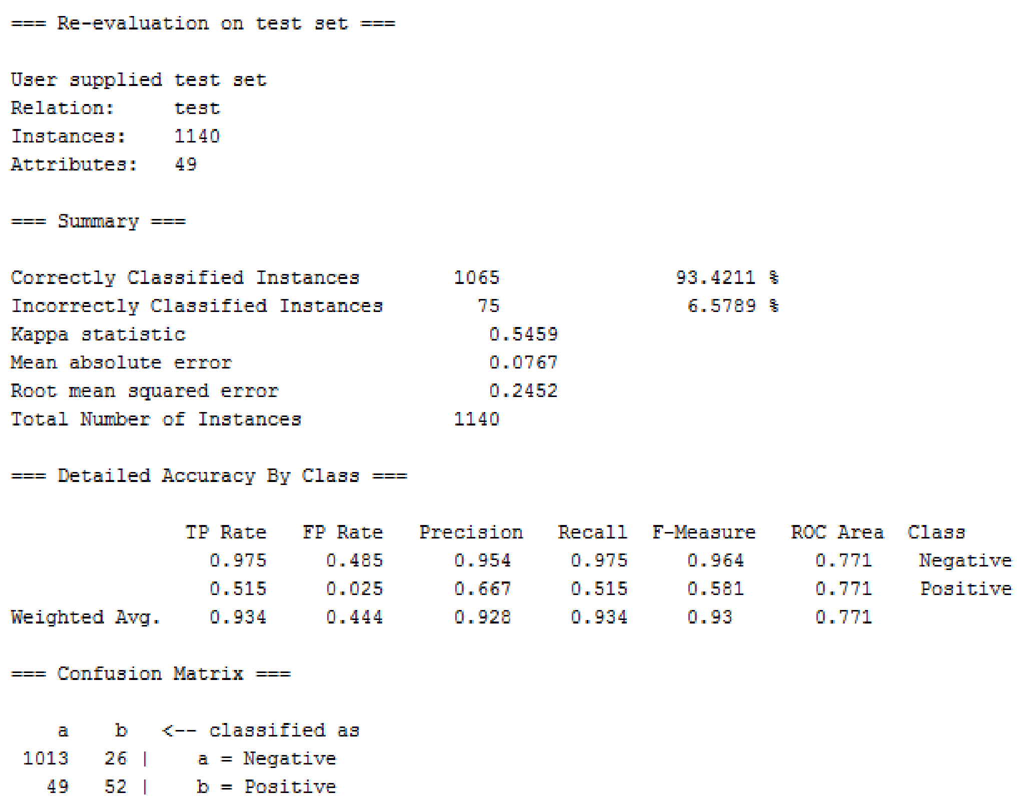


Figure 4.11: Threshold Selector ROC Curve for Negative class

4.5.4 C4.5 - Decision trees

When we give a data set to the C4.5 algorithm, it first makes an initial tree using the divide-conquer technique as bellow: If all the instances in data set belong to the same class, the tree will be a leaf labeled with the most frequent class I data set. Otherwise, select a test based on a single attribute with two or more outcomes. The algorithm will do this test the root of the tree with one branch for each outcome of test, divide data set to subsets according to the outcome for each case, and apply



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1065               93.4211 %
Incorrectly Classified Instances        75                6.5789 %
Kappa statistic                          0.5459
Mean absolute error                      0.0767
Root mean squared error                  0.2452
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.975     0.485      0.954     0.975     0.964      0.771    Negative
                 0.515     0.025      0.667     0.515     0.581      0.771    Positive
Weighted Avg.    0.934     0.444      0.928     0.934     0.93       0.771

=== Confusion Matrix ===

    a    b   <-- classified as
 1013   26 |    a = Negative
   49   52 |    b = Positive
```

Figure 4.12: J 48 algorithm

the same process recursively to each subset. In this algorithm attributes can be both kind of numerical and nominal and this determines the type and the format of the test outcomes. Then the initial tree is pruned to avoid over-fitting.

C4.8 which is improved version of C4.5 in Weka is knows as J48. So, the purpose is to learn the possibilities provided by the Weka program to build and visualize classification trees. In the classifier menu, select the J48 method from the trees sub-menu.

As we see in summary, algorithm has around 93% of accuracy with around 7% of error rate. Having a look at the confusion matrix shows that even algorithm just with 26 wrongly classified non singleton queries as singleton had good performance about them but with 49 instances of wrongly classified singleton instances had not good performance about them. Precision and recall are same also, algorithm have 0.95 about the negative class and 0.67 about positive class and also 0.97 for negative class in recall and 0.51 for positive class.

And last measure again ROC area in compare with other alorithms is not good with 0.77. It So, we can decide that J48 is not a good algorithm for our data set and models made based on this algorithm could not be useful for classification of queries.

4.5.5 ADTree

ADTrees were introduced by Yoav Freund and Llew Mason. However, the algorithm as presented had several typographical errors. Clarifications and optimizations were later presented by Bernhard Pfahringer, Geoffrey Holmes and Richard Kirkby. Implementations are available in Weka and JBoost. Original boosting algorithms typically used either decision stumps or decision trees as weak hypotheses. As an example, boosting decision stumps creates a set of T weighted decision stumps (where T is the number of boosting iterations), which then vote on the final classification according to their weights. Individual decision stumps are weighted according to their ability to classify the data.

Boosting a simple learner results in an unstructured set of T hypotheses, making it difficult to infer correlations between attributes. Alternating decision trees introduce structure to the set of hypotheses by requiring that they build off a hypothesis that was produced in an earlier iteration. The resulting set of hypotheses can be visualized in a tree based on the relationship between a hypothesis and its ”parent.”

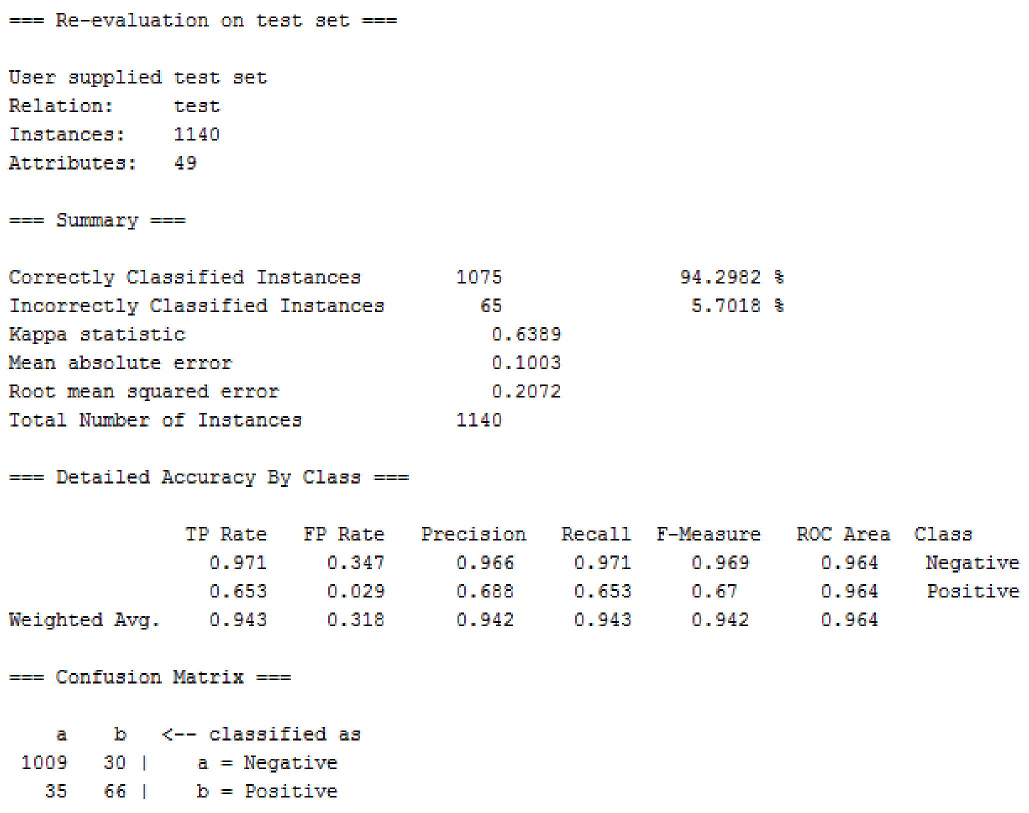
Another important feature of boosted algorithms is that the data is given a different distribution at each iteration. Instances that are misclassified are given a larger weight while accurately classified instances are given reduced weight.

After a weak performance with J48 algorithm let's try this ADTree algorithm on our data set and check. As we see down below the accuracy and error rate sounds good and model has 94% of correct classification and just 6% of error. But according to the ROC measure this algorithm among the tree classifiers seems more precise and performs well and does good job about the Positive class in compare with the J48 algorithm.

4.6 Imbalanced data and skewed distributions

4.6.1 What is a imbalanced data set

Several issues may impress output of a classifier algorithm. One of these issues depends in the distribution of instances of classes. For some kind of application domains, a huge disproportion in the number of cases belonging to each class is usual. For example, in detection of fraud in credit card transactions, the number of legitimate transactions is much higher than the number of fraudulent transactions. In insurance risk modeling, only a small percentage of the policyholders file one or more claims in any given time period. Also, in direct marketing, it is common to have a small response rate (about 1%) for most marketing campaigns. In our



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1075               94.2982 %
Incorrectly Classified Instances        65                5.7018 %
Kappa statistic                          0.6389
Mean absolute error                      0.1003
Root mean squared error                  0.2072
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.971     0.347      0.966     0.971     0.969      0.964    Negative
                 0.653     0.029      0.688     0.653     0.67       0.964    Positive
Weighted Avg.    0.943     0.318      0.942     0.943     0.942      0.964

=== Confusion Matrix ===

    a    b   <-- classified as
 1009   30 |    a = Negative
   35   66 |    b = Positive
```

Figure 4.13: ADTree Algorithm

problem we have more or less same condition. The number of instances for the Singletone class is around 10% of the total number of instances and multi tasks have the major part of data set with around 90% of total distribution. When this difference among the frequency of classes is large, the learning algorithm may have difficulties to learn how can realize the instances of minority class and final model will be weak about discovering it. In these kind of imbalanced and skewed data sets it easy for algorithm to put the instances in the class which has more probability than other one. Actually, this will not be a classifier based on the attributes and will be more about the population of the classes. I mean algorithm even can get a high classification accuracy even with putting all instances in the class with majority. Now we discuss several issues related to learning with skewed class distributions, such as the relationship between cost-sensitive learning and class distributions, and the limitations of accuracy and error rate to measure the performance of classifiers [18]. Most of learning algorithms are not able to make a model that accurately classifies the class with less frequency in skewed datasets. They normally have good accuracy and low error rate for the class with high population. Also in first glance the total accuracy and error rate of final model based on the imbalanced data set will be very satisfying but it is unacceptable for minority class.

The problem arises when the misclassification cost for the minority class is much higher than the misclassification cost for the majority class. Unfortunately, that is the norm for most applications with imbalanced data sets, since these applications aim to profile a small set of valuable entities that are spread in a large group of "uninteresting" entities. There are some works that have already done for solving this problem and in our work we will follow and use their suggested techniques[4, 14].

4.7 What is the solution

Researchers find out some solutions that have 2 main ways. First one does some changes in distribution and frequency of classes and second way does not change touch the data set and just adjust algorithm by applying cost matrix. Well, first technique tries to resample training set and modifies the distribution of instances of each class before than modeling and applying the classification algorithm. Re-sampling of training set could be done by increasing or decreasing the number of instances. Actually when the number of instances in a class is so higher than other one we decrease the distribution of majority class by removing some of the instances of that class and we actually do Under-sampling or we can increase the number of instances of minority class which is called over sampling. This way is not completely applicable without any effect on our final model but works very good and tackles the problem of imbalanced data. Over sampling can lead to over fitting and under-sampling sometimes outperforms oversampling but in other hands can lead to lose some useful information by removing instance. Next problem about the sampling technique is the strategy of sampling. For example how many and which of instances should be removed in under sampling technique or how many new instances and how should be created for oversampling. In cost matrix instances of minority class have high misclassification cost than the majority class.

Second technique as we mentioned before is cost sensitive classification which works by applying a cost matrix on classification algorithms which can be done by any algorithm. Cost-based learning can be divided into three steps; first step is making a specific learning algorithm sensitive to cost, second one is assigning examples to their lowest risk class, and converting an arbitrary learning algorithm to become cost sensitive. Adjusting the probability estimation or adjusting classification threshold can also help counter the problem. Finally and last step is building classifiers that learn each class separately can also be used to overcome the problem of imbalance. Most of these solutions have been discussed in this paper[18, 14]. DOwn below is the list of mentioned techniques for solving the problem of imbalanced data sets.

1. Under-sampling. One very direct way to solve the problem of learning from imbalanced data sets is to artificially balance the class distributions. Under-sampling aim to balance a data set by eliminating examples of the majority class;

2. Over-sampling. This method is similar to under-sampling. But it aims to achieve a more balanced class distributions by replicating examples of the minority class.

3. Assign misclassification costs. In a general way, mis-classify examples of the minority class is more costly than mis-classify examples of the majority class. The use of cost-sensitive learning systems might aid to solve the problem of learning from imbalanced data sets;

4.8 More Experiments

After assessing the mentioned ways about dealing with the imbalanced distributed data, now we are going back to work with Weka and try them. For the 2 first ways (Over-Sampling and Under-Sampling) there is a tool which is like a filter in Weka and we can apply it on the data set. In the filter section under the supervised subsection there is a filter in the name of SMOTE which is does what we want about oversampling and Under-sampling.

It re-samples a dataset by applying the Synthetic Minority Oversampling Technique (SMOTE). The original dataset must fit entirely in memory. The amount of SMOTE and number of nearest neighbors may be specified. For more information, see the paper "Synthetic Minority Over-sampling Technique"[3].

4.8.1 Options available in Weka or re-sampling

Class Value – The index of the class value to which SMOTE should be applied. Use a value of 0 to auto detect the non-empty minority class.

Nearest Neighbors – The number of nearest neighbors to use.

Percentage – The percentage of SMOTE instances to create.

Random Seed – The seed used for random sampling.

As our Yes class almost has the 10% of our class distribution, I apply 800% in SMOTE option for increasing the number of Yes class instances around 10 times.

In this way we will have a distribution which have around 50% of each class. Now, after adjusting all of setting we should apply the filter and result will be a new build dataset with a balanced class distribution which new made samples are generated from the 5 nearest instances.

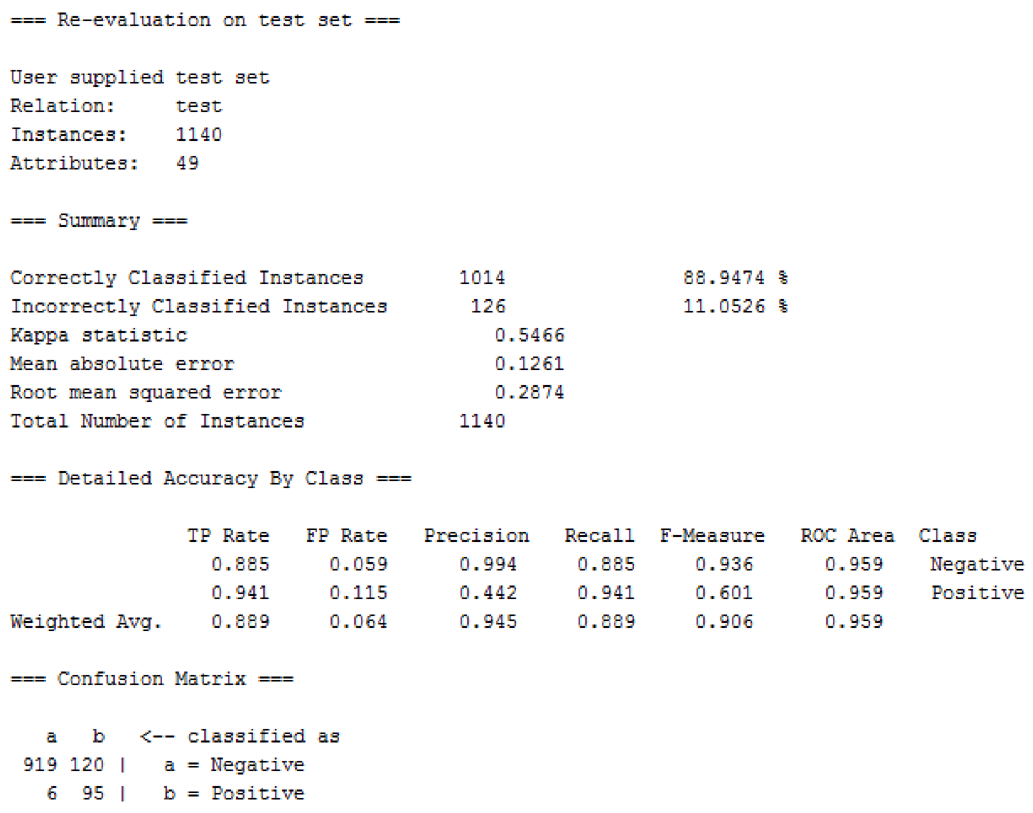
4.8.2 Modelling with new resampled traning data set

After all explained procedure of re-sampling our training set with some different options which were the way of creating new instances, now we have a new training set with 8208 instances and by chance exactly same size of two classes with 4104 instances in each. Just for making sure that new created instances won't be near each other we do a new shuffling with randomization that we had already used it before for initial training set. And after applying this filter is time to go to the classification process and try the previously tried algorithms with the initial training set.

Well, retraining the new new data set shows us some of classifier do give any considerable change in the result. As we mentioned before, by over sampling and under sampling actually we are going to training and make a model which will do same job about the test data which is not balance and is like the initial training set. Unfortunately, even they improved and made much more better model through the new re-sampled training data set but when we want to evaluate the model in last step they again give the same results as the have given with initial training set. And we did not get any improved results. Even some of them worked worse and the output model is working worse and give less accurate prediction. As we tried again this new re-sampled training set on different algorithms, also there are some algorithms which had bad result with the initial training set but this time with better results in learning and also testing procedures. Below we list some of the algorithms which performed better with new training set.

4.8.3 Logistic Regression with new data set

Logistic regression is an algorithm which in many cases looks like the ordinary regression. The procedure of this algorithm is modeling the relationship between a dependent variable with one or more independent variables. This algorithm make the possibility of checking the fit of model as well as at the importunateness of the relationship that we are modeling. However, the underlying principle of binomial logistic regression, and its statistical calculation, are quite different to ordinary linear regression. While ordinary regression uses ordinary least squares to find a best



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1014               88.9474 %
Incorrectly Classified Instances       126               11.0526 %
Kappa statistic                          0.5466
Mean absolute error                      0.1261
Root mean squared error                  0.2874
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.885     0.059      0.994     0.885     0.936      0.959    Negative
                 0.941     0.115      0.442     0.941     0.601      0.959    Positive
Weighted Avg.    0.889     0.064      0.945     0.889     0.906      0.959

=== Confusion Matrix ===

   a   b   <-- classified as
 919 120 |   a = Negative
   6  95 |   b = Positive
```

Figure 4.14: Logistic Algorithm with resampled training set

fitting line, and comes up with coefficients that predict the change in the dependent variable for one unit change in the independent variable, logistic regression estimates the probability of an event occurring (e.g. the probability of a pupil continuing in education post 16). Logestic Regression algorithm with the normal training set had a weak performance on Positive class which had almost 50 percent wrongly classified for this class which was same percentage of error rate on test set. But with the new training set it seems that the algorithm learns better about the positive class and classifies it better than before but it is not costless and we loose something in Negative class classification and model treats weakly about the negative class which results as high error rate in Negative class. So, we can conclude that this algorithm actually does not make noticeable difference in error rate and precision of model and even if does good job in positive class, it works worse than before about the negative class.

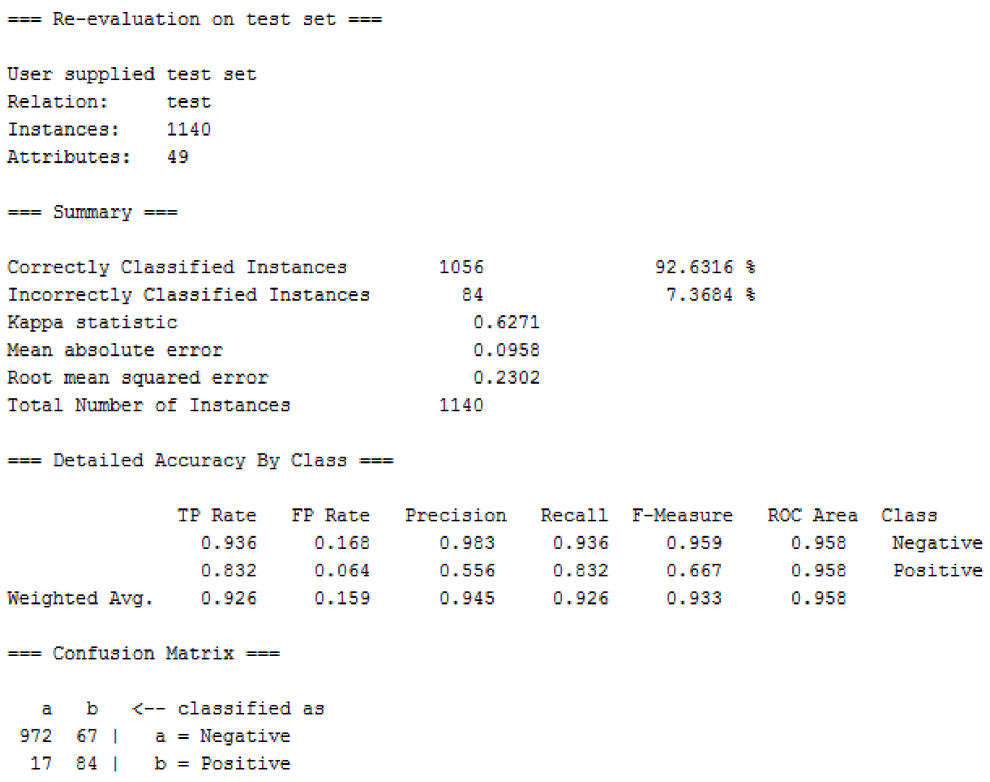
4.8.4 Bagging Algorithm with new data set

Bagging or boostrap aggregating, is a machine learning algorithm which has implemented for making better the accuracy of machine learning algorithms which are used in statistical classification. This algorithm can reduces variance and by this way avoids having over fitting in our models. This algorithm is usually is applied in decision trees but the point of this method is that it can be used with all type of the modelling algorithms.it is a special case of the model averaging approach. Given a standard training set D of size n, bagging generates m new training sets D_i, each of size n', by sampling from D uniformly and with replacement. By sampling with replacement, some observations may be repeated in each D_i. If n'=n, then for large n the set D_i is expected to have the fraction $(1 - 1/e)(\approx 63.2\%)$ of the unique examples of D, the rest being duplicates. This kind of sample is known as a bootstrap sample. The m models are fitted using the above m bootstrap samples and combined by averaging the output (for regression) or voting (for classification).

For bagging algorithm we also have almost same condition with the previous one. In total the numbers of correctly classified instances does not change significantly and the only thing which change is the balance of correctly classified for each of classes. In Positive class new model gets better in correctly classification and in negative class gets worse. But, even the total number of correctly classified instanced does not change a lot we reach to our goal. Because as we told before in imbalanced data we should take care about the cost of the classes. I mean if classifying singleton queries correctly is more important for us than classifying multi task queries, we have reached to our goal by this way and most of instances are classified correctly. Like the example of fraud in a bank system classifying a normal transaction as a fraud is not costly as classifying the fraud transaction as a normal one.

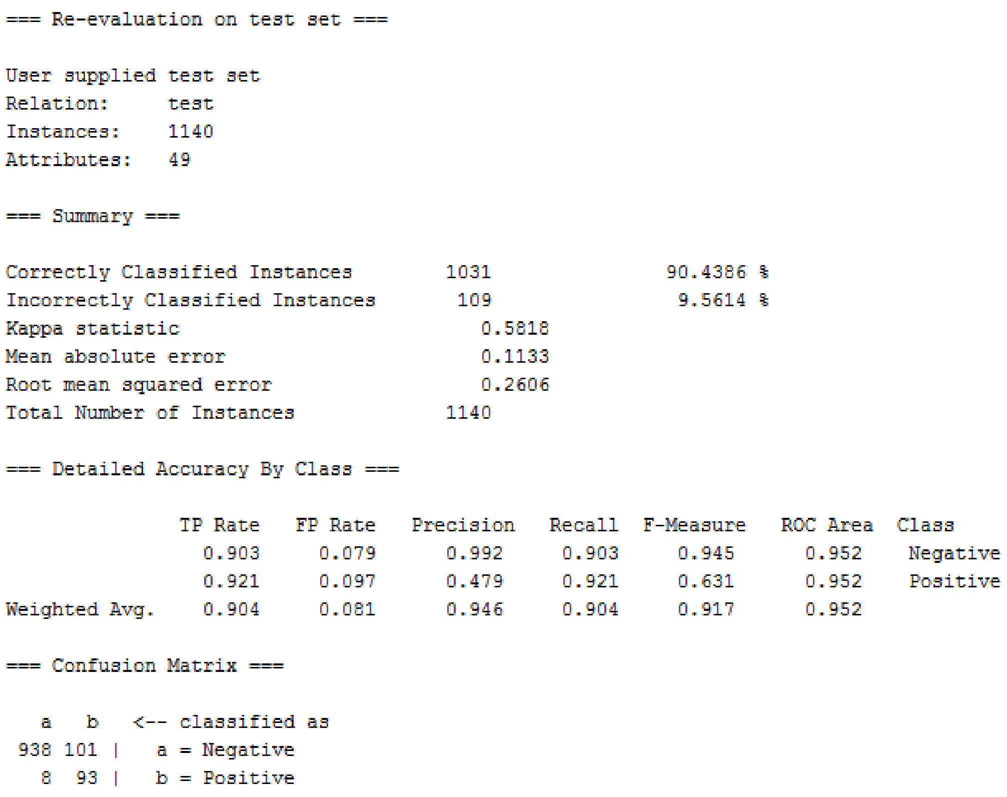
4.8.5 LoogitBoost Algorithm with new train set

This algorithm was one of our chose one as a good classifier for our problem and data set. Now with the new training set we get a model which in practice even give less number of correctly classified instances but as mentioned before this model is also learns better about the class which has more cost than other. This algorithm is one of the best with around 5 % of error rate about positive class has the best performance about the Singleton query discovering, even it does more mistake and mis-classify more multi task query in compare with the model made by the initial training set but in a case that purpose is discovering as much as possible of positive class absolutely this model will be one of the best.



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1056               92.6316 %
Incorrectly Classified Instances        84                7.3684 %
Kappa statistic                          0.6271
Mean absolute error                      0.0958
Root mean squared error                  0.2302
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.936     0.168      0.983     0.936     0.959      0.958    Negative
                 0.832     0.064      0.556     0.832     0.667      0.958    Positive
Weighted Avg.    0.926     0.159      0.945     0.926     0.933      0.958

=== Confusion Matrix ===

   a   b   <-- classified as
 972  67 |   a = Negative
  17  84 |   b = Positive
```

Figure 4.15: Bagging Algorithm with over sampled training set



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1031               90.4386 %
Incorrectly Classified Instances       109                9.5614 %
Kappa statistic                          0.5818
Mean absolute error                      0.1133
Root mean squared error                  0.2606
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.903     0.079      0.992     0.903     0.945      0.952    Negative
                 0.921     0.097      0.479     0.921     0.631      0.952    Positive
Weighted Avg.    0.904     0.081      0.946     0.904     0.917      0.952

=== Confusion Matrix ===

   a   b   <-- classified as
 938 101 |   a = Negative
   8  93 |   b = Positive
```

Figure 4.16: RegitBoost Algorithm with oversampled training set

4.9 Under-sampling

Vice versa, when we do under-sampling we will decrees the number of instances of class with high population to have a balanced number of instances. In our case we decrease around 800% of instances of negative class and then we will have less in compare with the original one (around 1000 instances) but the new one will have the balanced distribution with 50% of each class. After trying the over sampling method we can see that also in this way we can get some same results like oversampling. But there is a problem about under-sampling that we are not that much interested to do this technique and it is the loosing information. Actually, a good model from an algorithm in supervised learning could be built if we can supply enough instances for our algorithm and it can learn from each of instances. So, loosing each sample meas loosing some information and that leads a weak and not well trained model.

4.10 Cost sensitive classification in output

We talked about the imbalanced data and the ways of handling the problem of population of instances. We introduced the Over-sampling and Under-sampling method which were trying to balance the number of instances before training and making model. More precisely we should say that when we use these 2 method for solving the problem of imbalanced data actually we try to fix in in the first step, I mean we make a model based which is trained by balanced data set and the output is a model which take care about the class which has more cost than the others. Basically, when we re-ample a dataset with imbalanced instanced actually we indirectly say that the class with minority population s more important for us and any wrong classification about this class will be more costly for us. So, by training aa algorithm with re-sampled data set we say to our model that take care more about this class and the model is made by this consideration.

But, there is another way for handling the imbalanced data set and it works in different procedure of these two re-sampling way. The thid way that we explained before about imbalanced data is assigning misclassification costs. In this way we first learn from the original train set without balancing the number of instances and for the algorithm which are defining probability for classification, at the end and after modeling for final results made model consider the cost of misclassification and changes the threshold of the probability between two classes and puts even instances with less probability of being in class with minority population even it can be lead for misclassification for the other class.

For doing this in Weka by choosing the "CostSensetiveClassifier" we can try this way of tacking the imbalanced data. Actually it does not choose the classifier algorithm and we should decide about the algorithm which we want to use. Cost matrix is the matrix we should define and decide about the cost that we want to apply. Dimension of matrix depends on the class attribute that we have. As we have two class which are Positive and Negative, we will make a 2 dimension cost matrix that defines the cost of misclassification for each class by allocating the coast in non diagonal elements of matrix. Well, this cost normally defines according to the different parameters but in our case as we want to have reasonable misclassification in both classes according to the minority class which has around 10% of distribution we de try some different cost matrix. For example we try misclassification in Positive class 5 times costly than Negative class and some other random ones.

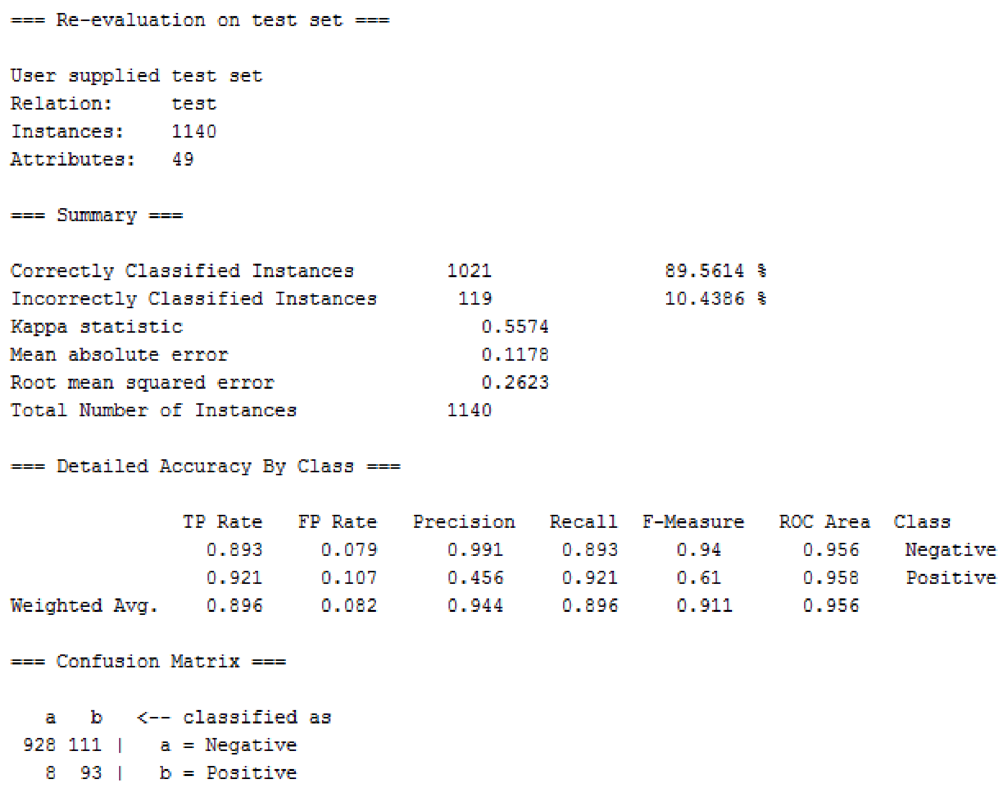
Now, according to our features type we will choose our classification algorithm. Based on previous experiment we can decide about the algorithms which had good results in initial classification.

4.10.1 Logestic Algorithm withwith cost sensitive classification

As we can see below, with cost matrix which in misclassification of positive class costs 5 time of a misclassification of negative class, and re-evaluating the produced model we can get more or less same results that we got in experiment that we did by training the Logistic algorithm based on the re-sampled training set. The precision of Positive class decrees in compare with he model which is made by initial training se and that is because of making more mistake in Negative class but the number of truly positive classified instances has been increased. ROC value and the area under the ROC value is also high and means that model doeas a good job for our imbalanced data set.

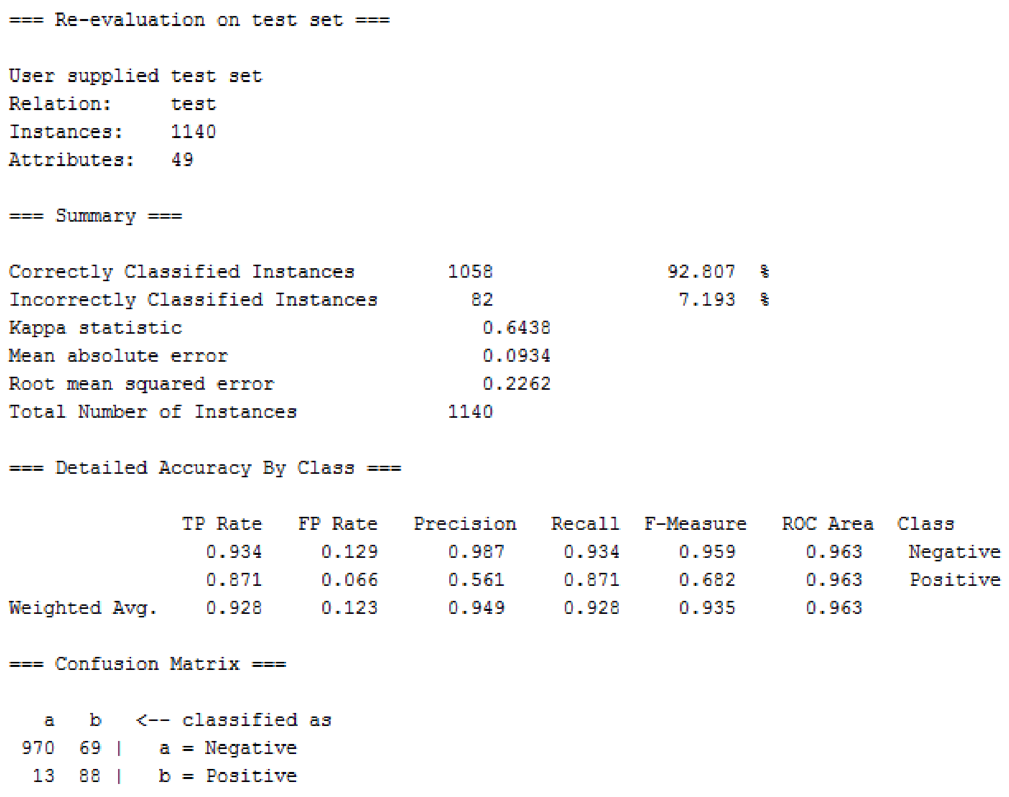
4.10.2 Bagging and LogitBoost Algorithms with cost sensitive classification

Applying these two algorithms with cost sensitive classification approves that more or less all do same job when we re-sample the training set by over or under sampling technique. Just in this technique we can adjust the cost matrix to have more desired results about the class which has more importance than the other one.



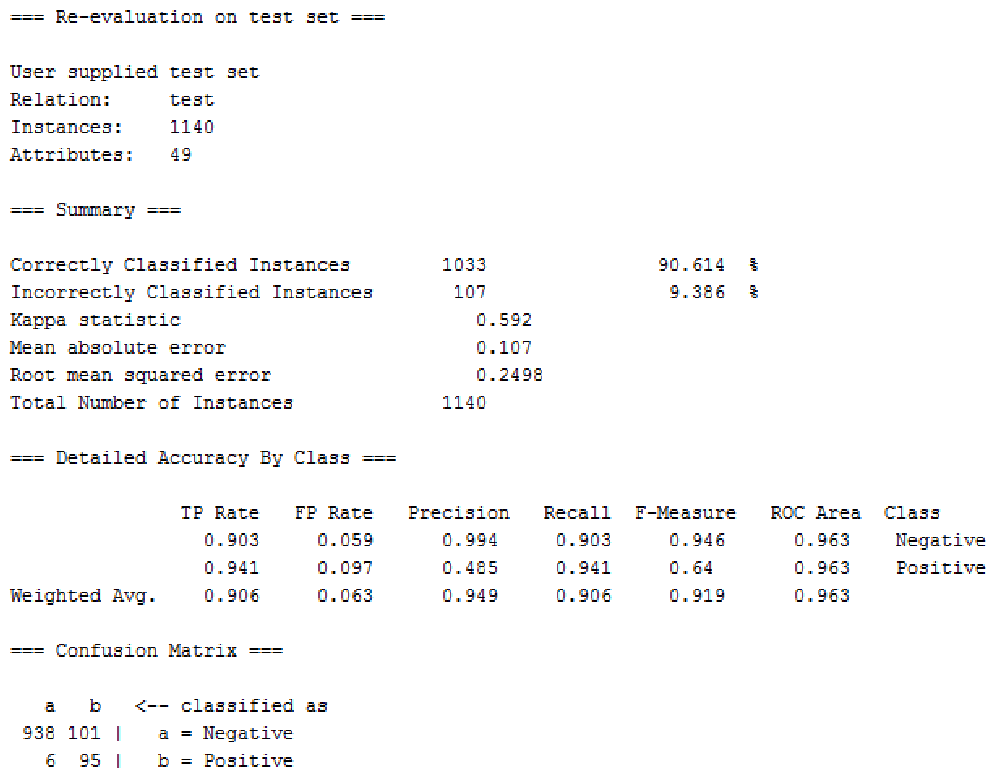
```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1021               89.5614 %
Incorrectly Classified Instances       119               10.4386 %
Kappa statistic                          0.5574
Mean absolute error                      0.1178
Root mean squared error                  0.2623
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.893     0.079      0.991     0.893     0.94       0.956    Negative
                 0.921     0.107      0.456     0.921     0.61       0.958    Positive
Weighted Avg.    0.896     0.082      0.944     0.896     0.911      0.956

=== Confusion Matrix ===

   a   b   <-- classified as
 928 111 |   a = Negative
   8  93 |   b = Positive
```

Figure 4.17: Logistic Algorithm wit cost sensitive classification



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1058               92.807  %
Incorrectly Classified Instances        82                7.193  %
Kappa statistic                          0.6438
Mean absolute error                      0.0934
Root mean squared error                  0.2262
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.934     0.129      0.987     0.934     0.959      0.963    Negative
                 0.871     0.066      0.561     0.871     0.682      0.963    Positive
Weighted Avg.    0.928     0.123      0.949     0.928     0.935      0.963

=== Confusion Matrix ===

   a   b   <-- classified as
 970  69 |   a = Negative
  13  88 |   b = Positive
```

Figure 4.18: Bagging Algorithm with cost sensitive classification



```
=== Re-evaluation on test set ===

User supplied test set
Relation:     test
Instances:    1140
Attributes:   49

=== Summary ===

Correctly Classified Instances        1033               90.614  %
Incorrectly Classified Instances       107                9.386  %
Kappa statistic                          0.592
Mean absolute error                      0.107
Root mean squared error                  0.2498
Total Number of Instances             1140

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.903     0.059      0.994     0.903     0.946      0.963    Negative
                 0.941     0.097      0.485     0.941     0.64       0.963    Positive
Weighted Avg.    0.906     0.063      0.949     0.906     0.919      0.963

=== Confusion Matrix ===

   a   b   <-- classified as
 938 101 |   a = Negative
   6  95 |   b = Positive
```

Figure 4.19: Logitboost Algorithm with cost sensitive classification

Chapter 5

Conclusion and Future work

5.1 Conclusion

In this work first of all we tried to define attributes as much as possible for each query. We used the query itself, time and number of clicks done for each query to define 48 lexical and semantic features. Then we had a new data set with 6383 instances and 49 attributes included the class attribute. According to the type of our attributes we tried to use more appropriate algorithms to make model for prediction. We came up with some algorithms which had good and acceptable enough for our purpose. For making sure and secure about the correctness of model actually we did 11 time of training and last final test. For each algorithm we did a cross validation with 10 fold and then when the results outcome from this step was good we shift to the last and 11th training with whole training set and made model controlled and test with allocated test set.

Another aspect that we discussed in this work was the issu of imbalanced data sets. We talked about the most common ways for dealing with the skewed data sets and used these techniques for even getting better results. Cost sensitive classification is the solution fro our case. But, there are two ways for applying the cost sensitive classification . One is directly applies during the learning procedures and the model which is made is actually cost sensitive. This technique needs preprocessing on training set and normally is done by Over and under-sampling and te algorithm get trained by the re-sampled data set. And other technique which works a bit in different way and model is made by original training data without any re-sampling and preprocessing and normally os done by algorithms that define probability for instances for classification. Then the prosecuted model applies the defined cost matrix when it wants to classify the instances based on the defined cost matrix and slightly changing the threshold of probability. The point of the cost sensitive classification is that, the total number of the correctly classified instances normally does not change a lot and just based on the importance each of our classes model tries to discover as much as possible instance of the class which is more costly than the other but it will increase the error rate of other class with less cost.

Now, according based on the our goal and application we can decide about the type of our modeling procedure. If we want to have a reasonable and fair classification among all instances with out applying any importunateness and cost we can use the mentioned algorithms that we applied on initial and original data set. But, if in a special application according to our need goal becomes classifying a special class, we can use the cost sensitive classification and put more cost in cost matrix for the class which we want to find as much as possible.

5.2 Future work

For this work we defined 48 attributes for every query. These are defined base on the query itself, time of doing query in a session and number of clicks done for each query in session. We know that all of these feature do not have same effect in classification. Absolutely some of them play more role in model for making decision about the type of class of each query. One phase of next step could be evaluating these defined queries and even pruning the less effective ones and do modeling without those one to compare the results based on remaining attribute. Another phase of our work could be adding some more and new attribute to improve the accuracy of defined models. We are tying to implement some frameworks like yoga and calculate a semantic feature based on this framework for our instance and see the effect of the newly defined features on our results.

As we know there are so many other semantic measure which can be defined for evaluating the semantic similarity of two text, our plan is applying them for queries and using them as new feature. There will be some boundaries because of the shortness of queries but still there are more options to use the semantic similarity measure for our feature work and making our classification model more and more precise about finding the single task queries.

Since, our work is quite new, there are a lot of potential fields about discovering tasks of queries and specially, single task queries so we will do more research about the fields which our research could be useful in them. These field can be recommendation systems and other similar ones.

References

[1] Doug Beeferman and Adam Berger. Agglomerative clustering of a search engine query log. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 407–416. ACM, 2000.

[2] Soumen Chakrabarti, Martin Ester, Usama Fayyad, Johannes Gehrke, Jiawei Han, Shinichi Morishita, Gregory Piatetsky-Shapiro, and Wei Wang. Data mining curriculum: A proposal (version 0.91). 2004.

[3] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *arXiv preprint arXiv:1106.1813*, 2011.

[4] Yetian Chen. Learning classifiers from imbalanced, only positive and unlabeled data sets. *Department of Computer Science, Iowa State University*, 2009.

[5] Shui-Lung Chuang and Lee-Feng Chien. Enriching web taxonomies through subject categorization of query terms from search engine logs. *Decision Support Systems*, 35(1):113–127, 2003.

[6] Silviu Cucerzan and Eric Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *EMNLP*, volume 4, pages 293–300, 2004.

[7] Daniel Gayo-Avello. A survey on session detection methods in query logs and a proposal for future evaluation. *Information Sciences*, 179(12):1822–1843, 2009.

[8] Matthias Hagen, Jakob Gomoll, and Benno Stein. Improved cascade for search mission detection. In *Proceedings of ECIR 2012 Workshop on Information Retrieval over Query Sessions*.

[9] Matthias Hagen, Benno Stein, and Tino Rüb. Query session detection as a cascade. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 147–152. ACM, 2011.

[10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[11] Daqing He and Ayse Göker. Detecting session boundaries from web user logs. In *Proceedings of the BCS-IRSG 22nd annual colloquium on information retrieval research*, pages 57–66, 2000.

[12] Bernard J Jansen and Amanda Spink. How are we searching the world wide web? a comparison of nine search engine transaction logs. *Information Processing & Management*, 42(1):248–263, 2006.

[13] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[14] William Klement, Szymon Wilk, Wojtek Michaowski, and Stan Matwin. Dealing with severely imbalanced data. In *ICEC 2009 Workshop, PAKDD*. Citeseer, 2009.

[15] Tessa Lau and Eric Horvitz. Patterns of search: Analyzing and modeling web query refinement. *Courses and lectures-International Centre for Mechanical Sciences*, pages 119–128, 1999.

[16] Claudio Lucchese, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Gabriele Tolomei. Discovering tasks from search engine query logs. *ACM Transactions on Information Systems (TOIS)*, 31(3):14, 2013.

[17] Oded Maimon and Lior Rokach. Introduction to knowledge discovery in databases. In *Data Mining and Knowledge Discovery Handbook*, pages 1–17. Springer, 2005.

[18] Maria Carolina Monard and GEAPA Batista. Learning with skewed class distributions. *Advances in Logic, Artificial Intelligence and Robotics*, pages 173–180, 2002.

[19] Seda Ozmutlu, H Cenk Ozmutlu, and Amanda Spink. Multitasking web searching and implications for design. *Proceedings of the American Society for Information Science and Technology*, 40(1):416–421, 2003.

[20] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. In *ACm SIGIR Forum*, volume 33, pages 6–12. ACM, 1999.

[21] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.