

Design of Interactive Feature Space Construction Protocol



A thesis submitted in partial fulfillment of the requirements for the
degree of
Doctor of Philosophy (Ph.D)

By

Mehnaz Khan

**P.G Department of Computer Sciences
Faculty of Applied Sciences and Technology
University of Kashmir**

Under the Supervision of

Dr. S.M.K Quadri

in

Computer Science

May 2013



Department of Computer Sciences

University of Kashmir
Hazratbal, Srinagar-190006

DECLARATION

This is to certify that the thesis entitled “**Design of Interactive Feature Space Construction Protocol**” submitted by Mehnaz Khan in the *Department of Computer Sciences, University of Kashmir, Srinagar*, for the award of the degree of **Doctor of Philosophy in Computer Science**, is a record of an original research work carried out by her under my supervision and guidance. The thesis has fulfilled all the requirements as per the regulations of the University and in my opinion has reached the standards required for the submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

**Supervisor and Head
(Dr. S.M.K. Quadri)**

Department of Computer Sciences
University of Kashmir
Srinagar, 190 006

Dated: 20th May' 2013

DEDICATED To The Only Power.....

The Gracious and the most Merciful

ACKNOWLEDGEMENTS

At the top, all the praises and thanks be to Allah, the Almighty Creator, the Master of the life and the day of recompense. It is He who has been always actually guiding me and showing me the straight and right path towards the success.

I am heartily thankful to my supervisor, Dr. S.M.K. Quadri, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. It is his belief that kept my nerves on the move right till the end. If he hadn't encouraged me all the way, I would have given this up a long time ago.

I would like to acknowledge and extend my heartfelt gratitude to other faculty members of our department, especially Dr. Manzoor Ahmad Chachoo, for their kind assistance and valuable advice.

I am indebted to all my friends especially Shabia who supported me, gave honest comments to make my work better. They also stayed with me in the process and handled me and my, sometimes out of control emotions, so well in a manner that kept me going through the thick and thin of this study.

I express my special thanks to the technical staff of the Department of Computer Sciences for providing me the necessary facilities.

This thesis would not have been completed without the love and support of my family. I would like to thank my parents, Mr. Mohd Ashraf Khan and Mrs. Firdous Mahajan, for their unconditional love, emotional support and encouragement. I thank them for giving me the opportunity of education from the best institutions and for their support throughout my life. I would also like to thank my sister and brother-in-law, Rakhshanda and Zahoor Khan for their love and emotional support through these times and for giving me my nephew, Amaan, whose photo in my laptop made me smile in toughest times. How will I forget to thank my darling sister Misbah for her persistent love, encouragement and prayers?

I offer my regards and blessings to all who supported me in every respect during the successful realization of thesis, as well as expressing my apology that I could not mention them personally one by one.

Mehnaz Khan

Abstract

Machine learning deals with designing systems that learn from data i.e. automatically improve with experience. Systems gain experience by detecting patterns or regularities and using them for making predictions. These predictions are based on the properties that the system learns from the data. Thus when we say a machine learns, it means it has changed in a way that allows it to perform more efficiently than before. Machine learning is emerging as an important technology for solving a number of applications involving natural language processing applications, medical diagnosis, game playing or financial applications. Wide variety of machine learning approaches have been developed and used for a number of applications.

We first review the work done in the field of machine learning and analyze various concepts about machine learning that are applicable to the work presented in this thesis. Next we examine active machine learning for pipelining of an important natural language application i.e. information extraction, in which the task of prediction is carried out in different stages and the output of each stage serves as an input to the next stage.

A number of machine learning algorithms have been developed for different applications. However no single machine learning algorithm can be used appropriately for all learning problems. It is not possible to create a general learner for all problems because there are varied types of real world datasets that cannot be handled by a single learner. For this purpose an evaluation of the machine learning algorithms is needed. We present an experiment for the evaluation of various state-of-the-art machine learning algorithms using an interactive machine learning tool called WEKA (Waikato Environment for Knowledge Analysis). Evaluation is carried out with the purpose of finding an optimal solution for a real world learning problem-credit approval used in banks. It is a classification problem.

Finally, we present an approach of combining various learners with the aim of increasing their efficiency. We present two experiments that evaluate the machine learning algorithms for efficiency and compare their performance with the new combined approach, for the same classification problem. Later we show the effects of feature selection on the efficiency of our combined approach as well as on other machine learning techniques. The aim of this work is to analyze the techniques that increase the efficiency of the learners.

Contents

	Page No.
List of Figures	viii-ix
List of Tables	x
List of Symbols	xi
1. Introduction	1-7
1.1.Introduction	1
1.2.Approaches of machine learning	2
1.2.1. Supervised Machine Learning	2
1.2.2. Unsupervised Machine Learning	3
1.2.3. Semi-supervised Machine Learning	3
1.3. Costs involved in various machine learning strategies	4
1.4. Active Learning	5
1.5.Thesis Statement	5
1.6.Thesis Outline	7
2. Review of Literature	8-29
2.1.Review of Research Work in NLP	8
2.1.1. Theoretical developments in NLP	8
2.1.1.1. Statistical Methods	8
2.1.1.2. Use of WordNet for NLP research	9
2.1.1.3. Use of finite state methods in NLP	10
2.1.2. NLP Applications	11
2.1.2.1. Automatic Abstracting	11
2.1.2.2. Information Retrieval	12
2.1.3. NLP Interfaces	13
2.1.4. NLP Software	14
2.2.Review of Research Work in Machine Learning	15
2.2.1. Active Learning Scenarios	15
2.2.1.1. Membership Query Synthesis	16
2.2.1.2. Stream-based Selection/ Selective sampling	17
2.2.1.3. Pool-based Selection	18
2.2.2. Querying Strategies	19
2.2.2.1. Uncertainty Sampling	19
2.2.2.2. Query-By-Committee	20
2.2.2.3. Unreliability Sampling	21
2.2.2.4. Expected Model Change	22

2.2.2.5.	Estimated Error Reduction	22
2.2.2.6.	Density-Weighting Methods	23
2.2.3.	Structured Outputs	24
3.	Background	30-46
3.1.	Supervised Learning	30
3.1.1.	Terminology	31
3.1.2.	Version Space and Feature Space	33
3.1.3.	Supervised Machine Learning Procedure	35
3.1.4.	Examples of Supervised Machine Learning: Classification and Regression	36
3.2.	Machine Learning for Complex Problems	39
3.2.1.	Learning Structured Instances	40
3.2.2.	Learning Pipeline Models	43
3.3.	Pool-Based Active Learning	45
4.	Information Extraction and Machine Learning-A Pipelined Approach	47-61
4.1.	Introduction	47
4.1.1.	An Example of Pipelining	48
4.1.2.	Why Active Learning	49
4.2.	Simple Architecture of Information Extraction	50
4.3.	Pipelining and Machine Learning	53
4.4.	Stages of Information Extraction used in Pipelining	56
4.4.1.	Including POS Tagging in Pipelining	57
4.4.2.	Active learning for Entity and Relation Detection	59
4.5.	Evaluation Measures	59
5.	Evaluating Machine Learning Techniques for Efficiency	62-98
5.1.	Introduction	62
5.1.1.	WEKA- Interfaces	63
5.1.1.1.	Explorer	63
5.1.1.2.	Experimenter	65
5.1.1.3.	Knowledge Flow	66
5.1.2.	Datasets	67
5.1.2.1.	Preparing Datasets	67
5.1.2.2.	Training sets and Tests sets	69
5.1.2.3.	Using the training and test sets in WEKA	70
5.2.	Learning problem and the Dataset used in our experiments	73
5.2.1.	Understanding the problem	73
5.2.1.1.	Risk involved in credit approval	74
5.2.1.2.	Credit evaluation method	74

5.2.1.3. Automating the process	75
5.2.2. Description of the Dataset used	75
5.3.Learning Methods Chosen For Evaluation	77
5.3.1. ZeroR and OneR	78
5.3.2. NaiveBayes and NaiveBayesUpdateable	79
5.3.3. MultiLayer Perceptron	79
5.3.4. J48 and Random Forest	80
5.3.5. KStar (K*)	82
5.3.6. AdaBoostM1 and Bagging	82
5.4.Experimental Setup	83
5.4.1. Experimental Procedure	86
5.4.2. Experimental Results	86
5.5. Conclusion	97
6. A Combined Approach Towards Learning And Feature Design	99-124
6.1. Introduction	99
6.1.1. Why Python	99
6.1.2. Python Machine Learning tool	100
6.2.Combined Learners	102
6.2.1. Types of Combination Techniques	102
6.2.2. Related Literature	107
6.3.Our approach towards combining learners	110
6.3.1. Procedure of our approach	111
6.3.2. Experimental Setup	112
6.3.3. Results	115
6.4.Feature Space Design	117
6.4.1. Feature Selection	119
6.4.2. Basic Steps in Feature Selection	120
6.4.3. Experiment and Results	121
7. Conclusion and Future Work	125-126
7.1. Conclusions	125
7.2. Future Work	125
Publications	127
References	128-149

List of Figures	Page No.
1.1. Active Learning Scenarios	6
3.1. Version Space	34
3.2. Entity and Relation detection from text	40
3.3. Information Extraction (a) as Sequence Labeling (b) as sequence model representing a finite state machine	41
3.4. Pipelined Named Entity Recognition	44
3.5. Pool-Based Active Learning	46
4.1. Pipelined Segmentation and Entity Detection	49
4.2. Simple Architecture of Information Extraction System	50
4.3. Tokenization and Labeling	51
4.4. Entity Detection	51
4.5. Process of Active Learning	54
4.6. A Venn diagram illustrating the relationship between actual and predicted positives	60
5.1. WEKA Explorer Interface showing Preprocess Tab	63
5.2. WEKA Explorer Interface showing Classify Tab	64
5.3. WEKA Experimenter Interface	66
5.4. Data in Excel spreadsheet	68
5.5. Data after loading in MS Word	68
5.6. Data after adding tags	69
5.7. Loading Dataset from URL	71
5.8. Using the Randomize filter	71
5.9. Using RemovePercentage filter	72
5.10. Using RemovePercentage filter with invertSelection	73
5.11. Credit Dataset	84
5.12. trainingcredit.arff file loaded in WEKA	85
5.13. testingcredit.arff file loaded in WEKA	85
5.14. Results of J48 on trainingcredit.arff	86
5.15. Results of J48 on testingcredit.arff	87
5.16. Results of RandomForest on testingcredit.arff	88
5.17. Results of ZeroR on testingcredit.arff	89

5.18.	Results of OneR on testingcredit.arff	90
5.19.	Results of NaiveBayes on testingcredit.arff	91
5.20.	Results of NaiveBayesUpdateable on testingcredit.arff	92
5.21.	Results of AdaBoostM1 on testingcredit.arff	93
5.22.	Results of Bagging on testingcredit.arff	94
5.23.	Results of MultiLayerPerceptron on testingcredit.arff	95
5.24.	Results of KStar on testingcredit.arff	96
5.25.	Time chart of algorithms	97
5.26.	Comparison of Algorithms By Percentage Of Correct Instances	97
6.1.	Bagging	104
6.2.	Boosting	105
6.3.	Stacking	106
6.4.	Cascading	106
6.5.	Flow of the Combined Technique	112
6.6.	Running a Script in Interactive Window in Python	114
6.7.	Results of our script on “testingcredit” file	115
6.8.	Comparison on the basis of Classification Accuracy	116
6.9.	Comparison on the basis of F-Measure	117
6.10.	Results of feature subset selection on “testingcredit” with margin 0.010	122
6.11.	Results of feature subset selection on “testingcredit” with margin 0.020	123

List of Tables

	Page No.
1.1. Instances with known labels	2
5.1. Example of a Dataset	67
5.2. Australian Credit Approval Dataset	76
5.3. Class Distribution	76
5.4. Comparison of algorithms	98
6.1. Comparison of learners	116
6.2. Before and after feature selection comparison of learners with margin 0.010	122
6.3. Before and after feature selection comparison of learners with margin 0.020	124
6.4. Comparing F-Measure at different margins	124

List of Symbols

x	Input item
y	Output label
f	Mapping function
h	Hypothesis
X	Input Space
\mathbf{X}	Input Feature Vector Space
Y	Output Space
\hat{y}	Predicted Output
w	Output Label Value
D	Probability Distribution
Φ	Feature Vector Generating Procedure
R	Real Number
\mathbf{x}	Input Feature Vector
H	Hypothesis Space
S	Data Sample
\hat{h}	Learned Hypothesis
F	Hypothesis Scoring Function Space
ρ	Margin of an Instance
L	Loss Function
\hat{f}	Learned Mapping Function
A	Learning Algorithm
U	Unlabeled Data Source
L	Labeled Data Source
q	Querying function

Chapter 1
Introduction

1.1. Introduction

Artificial Intelligence (AI) is the branch of computer science which deals with the study and creation of intelligent machines where an intelligent machine is a system which shows some form of intelligence i.e. a system which is capable of taking actions by observing its environment. These systems are capable of mimicking the human mind, understanding speech, and so on. In other words, an intelligent machine is a machine that can “think”. Natural Language Processing (NLP) is a field of artificial intelligence that is concerned with the interactions between the computers and the natural languages used by humans. NLP provides a method of human-computer interaction. It is concerned with interfacing computer representations of information with natural languages used by humans. It deals with examining the use of computers in understanding and manipulating the natural language text and speech. In the field of NLP, the aim of the researchers is to observe and collect the necessary information regarding how different natural languages are being used and understood by humans. This information is then used by the researchers for developing the tools for making the computers understand and manipulate the natural languages to perform desired tasks.

Some of the important natural language processing tasks include parsing, machine translation, information extraction, automatic abstracting, information retrieval, part-of-speech tagging, and question answering and so on. These days machine learning has emerged as an important technology for solving all these NLP tasks. Before the use of machine learning approaches, NLP tasks were implemented directly by hand coded set of rules. The machine learning algorithms automatically learn such rules by analyzing a large set of corpora (singular, “corpus”). A corpus is a collection of individual sentences or documents that have been hand annotated with the correct values to be learned. These corpus-based techniques have emerged as the dominant paradigm for NLP tasks.

The work in this thesis revolves around applying machine learning techniques for solving various issues. Mainly we have focused on an NLP problem and a real world financial application. A number of different types of machine learning algorithms have been used to solve these tasks. Some of the types include supervised learning, unsupervised learning, and semi-supervised learning.

1.2. Approaches of machine learning

Some of the types of machine learning approaches discussed briefly here include supervised learning, unsupervised learning, and semi-supervised learning.

1.2.1. Supervised machine learning

Supervised learning [Kotsiantis, 2007] is a type of machine learning in which the algorithms are provided with the data instances and they produce hypothesis from that data that helps in prediction. In supervised machine learning a function is deduced from the supervised data. Supervised learning is a process in which the task of the function is to predict the correct output from the inputs. This is done by deciding to which of the classes the new input belongs. The algorithm decides this by analyzing the data that is provided to it i.e. the training data. It consists of labeled instances i.e. inputs as well as their output classes. The task of supervised learning algorithms is to analyze the training data and produce a function. If the output of the function is discrete then it is called a classifier and if it is continuous then it is called a regression function. The inferred function should be capable of predicting the correct output value for any valid input. For doing this the learning algorithm must be able to generalize from the training data to unseen situations in a reasonable way. Supervised learning is the learning based on training data. Machine learning algorithms use the datasets that consist of a number of instances that are represented using the same set of features. Supervised learning differs from unsupervised learning in that it consists of the instances that have known labels (the corresponding correct outputs), whereas in unsupervised learning instances are unlabeled. Table 1.1[Kotsiantis, 2007] shows instances with known labels.

Table 1.1: Instances with known labels

Case	Feature 1	Feature 2	...	Feature n	Class
1	xxx	x		xx	good
2	xxx	x		xx	good
3	xxx	x		xx	bad
...					...

1.2.2. Unsupervised machine learning

Unsupervised learning [Duda *et al.*, 2001; Hinton and Sejnowski, 1999; Ghahramani, 2004] is different from supervised learning. In unsupervised learning the dataset consists of instances that are not labeled. In this learner is given only unlabeled examples. As already discussed, in supervised learning algorithms mapping is carried out from the input to an output and the correct values of the output i.e. known label are provided by a supervisor. In contrast, the unsupervised learning algorithms do not have any supervisor but only have input data. The goal of unsupervised learning is finding out the regularities in the input [Alpaydin, 2010]. The aim of unsupervised learning is determining the organization of the data. Density estimation is one of the examples of unsupervised machine learning. An important method of density estimation is clustering whose task is to find the clusters or groupings of input.

Consider a machine (or living organism) which receives some sequence of inputs. Let x_1 , x_2 , x_3 and so on, represent some sequence of inputs received by some machine. This input is often referred to as data. In supervised learning the machine is also provided with a sequence of desired outputs y_1 , y_2 , y_3 and so on, and the aim of the machine is to learn to generate the correct output for a new input. In case of classification the output can be a class label and in case of regression the output can be a real number. However, in unsupervised learning the machine simply receives inputs x_1, x_2, \dots , but does not receive the supervised target outputs [Ghahramani, 2004].

1.2.3. Semi-supervised machine learning

Semi-supervised learning algorithms use both labeled/annotated and unlabeled data in contrast to supervised learning where the data is all labeled and unsupervised learning in which the data is all unlabeled. Semi-supervised learning algorithm is provided with a small amount of labeled data and a large amount of unlabeled data.

In supervised machine learning the algorithms use only labeled data or the supervised data (i.e. feature/label pairs). However, it is difficult to obtain the labeled data. Because obtaining labeled data is a time consuming and expensive process as it needs the work of many experienced human annotators. As opposed to it, it is easy to collect the unlabeled data, but there are only a few ways of using them. So in order to get rid of this problem, semi-supervised learning techniques are used. These techniques show a lot of improvement in learning accuracy by using large amount of unlabeled data, together with the labeled data, to build better classifiers. Semi-supervised

learning techniques are of great use as they provide high accuracy as well as reduce human labor [Zhu, 2008].

Semi-supervised learning can be either transductive or inductive. In transductive learning the algorithm works only on the labeled and unlabeled training data, and cannot handle unseen data. However, inductive learners in contrast to transductive learners can naturally handle unseen data. Moreover, in semi-supervised classification, the learner has additional unlabeled data and the aim is classification and in semi-supervised clustering, the learner has unlabeled data with some pair wise constraints and the aim is clustering.

1.3. Costs involved in various machine learning strategies

The use of machine learning techniques in solving a number of problems in various fields has increased rapidly. These techniques are being widely accepted and implemented. This has led the researchers and developers to show a considerable amount of interest in minimizing the costs involved in using these techniques and developing such systems. For the successful implementation of machine learning techniques, significant amount of effort and cost is involved because of obtaining large labeled data sets and feature engineering. These problems get more intensified when the systems are implemented over wide range of data.

As discussed earlier, supervised machine learning techniques are quite expensive as they require obtaining large amounts of annotated data. Hence a lot of research work has been carried out regarding reducing labeled data requirements. On the other hand, unsupervised learning makes use of only unlabeled data, hence reducing the labeling costs involved. But unsupervised learning is often not directly applicable. Therefore another strategy that is used pre-clusters the data and only requires labels from representative points [Nguyen and Smeulders, 2004]. As discussed before, in between the two extremes i.e. supervised and unsupervised learning lies semi-supervised learning, where the learning algorithm is provided with a small amount of labeled data and a large amount of unlabeled data. Some of the commonly used approaches of semi-supervised learning are transductive learning [Joachims, 1999], bootstrapping [Abney, 2002], co-training [Blum and Mitchell, 1998], expectation-maximization (EM) algorithm, and graph-based methods. Another learning technique that has been used that minimizes the annotation costs is domain adaptation [Blitzer, 2008; Jiang, 2008]. In this technique learners are trained on a source distribution and modified using a small amount of data from a target

distribution. Human computation [Ahn, 2005] is a learning technique in which the annotation task is framed in such a way that annotators label data unknowingly.

The machine learning technique for reducing labeling costs studied in this thesis is active learning [Settles, 2010].

1.4. Active Learning

Like semi-supervised learning, active learning algorithms also work with small set of labeled data and a large set of unlabeled data. However, in active learning the learning algorithm is capable of selecting additional instances to be labeled by maintaining access to the annotator. Thus active learning provides a way to reduce the labeling costs by labeling only the most useful instances for learning. Active learning reduces the amount of user effort required to learn a concept by reducing the number of labeled examples required [Arora and Agarwal, 2007].

In this learning technique, the learner is responsible for actively participating in the collection of the training examples i.e. obtaining the training set. The learner is capable of selecting a new input, observing the resulting output and including the new example based on the input and output into its training set. An important question that arises here is how to choose which input to try next [Cohn *et al.*, 1996]. The learner uses some strategies for choosing the examples. The examples are chosen by making queries to the expert. The query strategy frameworks that have been used are uncertainty sampling [Lewis and Gale, 1994] and query-by-committee [Seunget *al.*,1992]. These strategies will be discussed in the later chapters.

There are different circumstances in which the learner may be able to ask queries. The learner may construct its own examples (membership query synthesis), request certain types of examples (pool-based sampling), or determine which of the unlabeled examples to query and which to discard (selective sampling). These are shown in Figure 1.1 [Settles, 2010].

In active learning, the learner examines the unlabeled data and then queries only for the labels of instances which it considers to be informative. Therefore, an active learner learns only what it needs to in order to improve, thus reducing the overall cost of training an accurate system.

1.5. Thesis Statement

This thesis aims to explore various machine learning protocols. This work examines the applicability of various machine learning techniques to complex problems with respect to the

natural language processing applications. The chapters that follow review the research work carried out in the field of machine learning and discuss the developments and applications of NLP, describe various types of machine learning approaches and concepts relevant to the work presented in this thesis, examine active learning with respect to information extraction using pipelining, show the performance evaluation of various state-of-the-art machine learning algorithms using an interactive machine learning tool, WEKA, on a real world problem using a real world dataset, and finally present a combined approach for the design of a learner that shows an increase in the efficiency of classification tasks of machine learning.

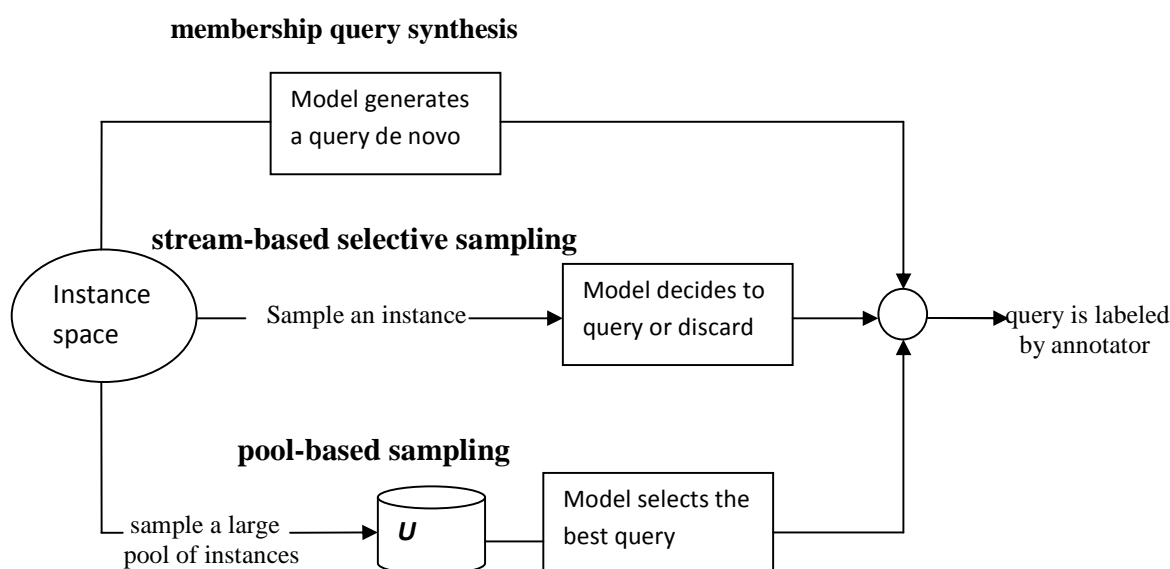


Figure 1.1: Active Learning Scenarios

The hypotheses supported in this thesis are:

- i. Machine learning strategies that take into consideration the informativeness or the relevance of instances can perform better with fewer labeled examples as compared to other learning approaches.
- ii. Active learning strategies reduce the costs of learning systems which actively participate in the collection of examples by maintaining access to the annotator.
- iii. Machine learning algorithms perform more efficiently for a classification task when they are combined together. For the prediction of the correct output class, combined

learner selects the class to which highest probability has been assigned among all the learners.

1.6. Thesis Outline

The remainder of this thesis is organized as follows:

- Chapter 2 presents a detailed study of the work done in the field of machine learning and NLP. It discusses the related literature along several dimensions. It presents the theoretical developments and applications of NLP.
- Chapter 3 discusses the basic concepts about machine learning that are relevant to the work presented in this thesis. It discusses supervised and active machine learning, learning structured instances and pipeline models.
- Chapter 4 discusses the use of machine learning for an important natural language application i.e. information extraction. It examines a pipelined approach for information extraction with respect to machine learning.
- Chapter 5 presents an evaluation of state-of-the-art machine learning algorithms on the basis of efficiency, for the task of classification. It begins by providing important concepts about WEKA- a tool for machine learning, and the process of preparing datasets. Later it presents the experiment and discusses the results.
- Chapter 6 presents a combined approach for the design of a learner that aims at increasing the efficiency of the learning tasks. It begins by providing the procedure of the combined approach and later presents the experiment and the results. In the second part of the chapter, we show the effect of feature selection on our combined approach and present its experiment and compare the results.
- Chapter 7 summarizes the primary contributions of this work and also presents the future directions of our work and in active learning.

Chapter 2
Review of Literature

This chapter describes the research literature relevant to the primary aspects of this thesis. The core aspects of this thesis are machine learning applications to natural language processing and classification techniques. Both these fields have received a lot of attention in the past years and there are a number of popular texts with relevant background material [Duda *et al.*, 2001; Russell and Norvig, 2003; Manning and Schutze, 1999; Jurafsky and Martin, 2008]. As there is an enormous amount of literature available on both these aspects, these works can be described along several dimensions.

2.1. Review of Research Work in NLP

Natural Language Processing (NLP) is that field of computer science which consists of interfacing computer representations of information with natural languages used by humans. It examines the use of computers in understanding and manipulating the natural language text and speech. Over the past years, a lot of research has been done in the field of NLP. Some of the recent works have been discussed here. Kumarana *et al.* (2011) have developed a multilingual content creation tool for Wikipedia. Optimal Search for Minimum Error Rate Training has been discussed by Michel and Chris (2011). Associating Web Queries with Strongly-Typed Entities [Patrick *et al.*, 2011], Linguistic Style Accommodation in Social Media [Cristian *et al.*, 2011], Predicting the Importance of Newsfeed Posts and Social Network Friends [Tim *et al.*, 2010], Wiki BABEL: A System for Multilingual Wikipedia Content [Kumaran *et al.*, 2010], The utility of article and preposition error correction systems for English language learners: Feedback and Assessment [Martin *et al.*, 2010]. The work presented in this Section has been previously published [Khan, Dar and Quadri, 2012].

2.1.1. Theoretical developments in NLP

Theoretical developments in NLP can be grouped into following classes: (i) statistical and corpus-based methods in NLP, (ii) use of WordNet for NLP research, (iii) use of finite-state methods in NLP.

2.1.1.1. Statistical Methods

The models and methods used in solving NLP problems are broadly classified into two types: deterministic and stochastic. A mathematical model is called deterministic if it does not involve

the concept of probability; otherwise it is said to be stochastic. A stochastic model can be probabilistic or statistical, if its representation is from the theories of probability or statistics, respectively [Edmundson, 1968]. Statistical methods are used in NLP for a number of purposes, e.g., speech recognition, part-of-speech tagging, for generating grammars and parsing, word sense disambiguation, and so on. There has been a lot of research in these areas. Geoffrey Zweig and Patrick Nguyen (2009) have proposed a segmental conditional random field framework for large vocabulary continuous speech recognition [Geoffrey and Patrick 2009]. Gerasimos Potamianos, Chalapathy Neti, Ashutosh Garg, Guillaume Gravier and Andrew W. Senior (2003) have reviewed *Advances in the Automatic Recognition of Audio-Visual Speech* and have presented the algorithms demonstrating that the visual modality improves automatic speech recognition over all conditions and data considered [Gerasimos *et al.*, 2003]. Raymond J. Mooney has developed a number of machine learning methods for introducing semantic parsers by training on a corpus of sentences paired with their meaning representations in a specified formal language [Raymond, 2007]. Marine CARPUAT and Dekai WU (2007) have shown that statistical machine translation can be improved by using word sense disambiguation. They have shown that if the predictions of the word sense disambiguation system are incorporated within a statistical machine translation model then the translation quality is consistently improved [Marine and Dekai, 2007].

2.1.1.2. Use of WordNet for NLP research

Mihalcea & Moldovan (1999) have proposed the use of WordNet to make the outcome of statistical analysis of natural language texts better. WordNet or the electronic dictionary is developed at Princeton University. It is a large database that serves as an important NLP tool consisting of nouns, verbs, adjectives and adverbs. These are arranged in the form of synonym sets (synsets). Each set represents one underlying lexical concept. These sets are linked with each other by means of conceptual-semantic and lexical relations. There are different wordnets for about 50 different languages, but they are not complete like the original English WordNet [Gerard and Gerhard, 2009]. WordNet is now used in a number of NLP research and applications. One of the most important applications of WordNet in NLP is EuroWordNet developed in Europe. EuroWordNet is a multilingual database which consists of WordNets for the European languages. It has been structured in the same way as the WordNet for English. A

methodology for the automatic construction of a large-scale multilingual lexical database has been proposed where words of many languages are hierarchically organized in terms of their meanings and their semantic relations to other words. This database is capable of organizing over 800,000 words from over 200 languages, providing over 1.5 million links from words to word meanings. This universal wordnet has been derived from the Princeton WordNet. Lars Borin and Markus Forsberg have given a comparison between WordNet and SALDO. SALDO is a Swedish lexical resource which has been developed for language technology applications [Lars and Markus, 2009]. Japanese WordNet currently has 51,000 synsets with Japanese entries. Methods for enhancing or extending the Japanese Wordnet have been discussed. These include: increasing the cover, linking it to examples in corpora and linking it to other resources. In addition various plans have been outlined to make it more useful by adding Japanese definition sentences to each synset [Franciset *et al.*, 2009]. The use of WordNet in multimedia information retrieval has also been discussed and the use of external knowledge in a corpus with minimal textual information has been investigated. The original collection has been expanded with WordNet terms in order to enrich the information included in the corpus and the experiments have been carried out with original as well as expanded topics[Manuel *et al.*, 2011]. A Standardized Format for Wordnet Interoperability [Claudia *et al.*, 2009] has been given i.e., WordNet- LMF. The main aim of this format is to provide the WordNet with a format representation that will allow easier integration among resources sharing the same structure (i.e. other wordnets) and, more importantly, across resources with different theoretical and implementation approaches.

2.1.1.3. Use of finite state methods in NLP

The finite-state automation is the mathematical tool used to implement regular expressions – the standard notation for characterizing text sequences. Different applications of the Finite State methods in NLP have been discussed [Jurafsky and Martin, 2000; Kornai, 1999; Rocheand Shabes, 1997]. From past many years the finite state methods have been used in presenting various research studies on NLP. The FSMNLP workshops are the main forum of the Association for Computational Linguistics' (ACL) Special Interest Group on Finite-State Methods (SIGFSM)[Anssiet *et al.*, 2011].

2.1.2. NLP Applications

There are a number of applications of NLP e.g. machine translation, natural language text processing and summarization, user interfaces, multilingual and cross language information retrieval (CLIR), speech recognition, and expert systems, and so on. In this paper we discuss automatic abstracting and information retrieval.

2.1.2.1. Automatic Abstracting

Automatic abstracting or text summarization is a technique used to generate abstracts or summaries of texts. Due to the increase in the amount of online information, it becomes very important to develop the systems that can automatically summarize one or more documents [Dragomir *et al.*, 2002]. The main aim of summarization is to differentiate between the more informative or important parts of the document and the less ones [Dipanjan and Andre, 2007]. According to Radev *et al.* (2002) a summary can be defined as piece of text that can be produced from one or more texts in a way such that it conveys important information in the original text(s), and whose size is not more than half of the original text(s) and mostly significantly less than that". The summary can be of two types i.e. abstraction or extraction. Abstract summary is one in which the original documents' contents are paraphrased or generated, whereas in an extract summary, the content is preserved in its original form, i.e., sentences [Krystaet *al*, 2007]. Extracts are formed by using the same words, sentences of the input text, while abstracts are formed by regenerating the extracted content. Extraction is the process of identifying the important contents in the text while in abstraction the contents are regenerated in new terms. When the summaries are produced from a single document, it is called single document summarization. Multidocument summarization has been defined as a process of producing a single summary from a number of related documents. A lot of research has been done on automatic abstracting and text summarization. Zajicetal [David *et al.*, 2008] have presented single-document and multi-document summarization techniques for email threads using sentence compression. They have shown two approaches to email thread summarization i.e. Collective Message Summarization (CMS) and Individual Message Summarization(IMS). NeATS [Chin and Eduard, 2002] is a multidocument summarization system in which relevant or interesting portions about some topic are extracted from a set of documents and presented in coherent order. NetSum [Krystaet *al*, 2007] is an approach to automatic summarization based on

neural networks. Its aim is to obtain those features from each sentence which helps to identify its importance in the document. A text summarization model has been developed which is based on maximum coverage problem and its variant [Hiroya and Manabu, 2009]. In this some decoding algorithms have been explored such as a greedy algorithm with performance guarantee, a randomized algorithm, and a branch-and-bound method. A number of studies have been carried out on text summarization. An efficient linear time algorithm for calculating lexical chains has been developed for preparing automatic summarization of documents [Silber and McCoy, 2000]. A method of automatic abstracting has been proposed that integrates the advantages of both linguistic and statistical analysis. Jin and Dong-Yan (2000) have proposed a methodology for generating automatic abstracts that provides an integration of the advantages of methods based on linguistic analysis and those based on statistics [Songand Zhao, 2000].

2.1.2.2. Information Retrieval

Information retrieval (IR) is concerned with searching and retrieving documents, information within documents, and metadata about documents. It is also called document retrieval or text retrieval. IR concerns with retrieving documents that are necessary for the users' information. This process is carried out in two stages [Jun and Jianhan, 2009]. The first stage involves the calculation of the relevance between given user information need and the documents in the collection. In this stage probabilistic retrieval models that have been proposed and tested over decades are used for calculating the relevance to produce a "best guess" at a document's relevance. In the second stage the documents are ranked and presented to the user. In this stage the probability ranking principle (PRP) [Cooper, 1971] is used. According to this principle the system should rank documents in order of decreasing probability of relevance. By using this principle the overall effectiveness of an IR system maximizes.

There has been a lot of research in the field of information retrieval. Some of the recent developments are included here. ChengXiangZhai (2008) has given a critical review of statistical language models for information retrieval. He has systematically and critically reviewed the work in applying statistical language models to information retrieval, summarized their contributions, and pointed out outstanding challenges [ChengXiang, 2008]. Nicholas J. Belkin has identified and discussed few challenges for information retrieval research which come under the range of association with users [Nicholas, 2008]. An efficient document ranking algorithm

has been proposed that generalizes the well-known probability ranking principle (PRP) by considering both the uncertainty of relevance predictions and correlations between retrieved documents [Jun and Jianhan, 2009]. Michael *et al* have discussed the various problems, directions and future challenges of content-based music information retrieval [Michael *et al.*, 2008]. A unified framework has been proposed that combines the modeling of social annotations with the language modeling-based methods for information retrieval [Ding *et al.*, 2008].

2.1.3. NLP Interfaces

A natural language interface accepts commands in natural language and sends data to the system which then provides the appropriate responses to the commands. A natural language interface translates the natural language statements into appropriate actions for the system. A large number of natural language interfaces have been developed [Stock, 2000]. A number of question answering systems are now being developed that aim to provide answers to natural language questions, as opposed to documents containing information related to the question. These systems use a variety of IE and IR operations to get the correct answer from the source texts. In information retrieval and NLP, question answering (QA) is the task of automatically answering a question posed in natural language. To find the answer to a question, a QA computer program may use either a pre-structured database or a collection of natural language documents. Unlike information retrieval systems (Internet search engines), QA systems do not retrieve documents, but instead provide short, relevant answers located in small fragments of text. That is why QA systems are significantly slower and require more hardware resources than information retrieval systems [Surdeanu *et al.*, 2002]. QA track of TREC (Text Retrieval Conference) have shown some interesting results. Several steps were included in the technology used by the participants in the QA track. First, words like ‘who’, ‘when’ were identified to guess what was needed; and then a small portion of the document collection was retrieved using standard text retrieval technology. This was followed by a shallow parsing of the returned documents for identifying the entities required for an answer. If no appropriate answer type was found then best matching passage was retrieved. In TREC-8, the first QA track of TREC, the most accurate QA systems could answer more than 2/3 of the questions correctly [Voorhees, 1999]. In the second QA track (TREC-9), the best performing QA system, the Falcon system from Southern Methodist University, was able to answer 65% of the questions [Voorhees, 2000]. In the first two QA tracks

the questions were simple. In TREC 2001 QA track, which was the third running of a QA track in TREC, a number of conditions were included for increasing the practicality and complexity of the task [Ellen, 2001]. The TREC 2002 track repeated the main and list tasks from 2001, but with the major difference of requiring systems to return exact answers. The change to exact answers was motivated by the belief that a system's ability to recognize the precise extent of the answer is crucial to improving question answering technology [Ellen, 2002]. These runnings of QA track have been carried out every year till date by adding different conditions to make the QA tracks more realistic.

2.1.4. NLP Software

A number of NLP software packages and tools have been developed, some of which are available for free, while others are available commercially. These tools have been broadly classified into different types some of which are mentioned here. General Information Tools(e.g. Sourcebank – a search engine for programming resources., The Natural Language Software Registry), Taggers and Morphological Analyzers(e.g. A Perl/Tk text tagger, AUTASYS – which is a completely automatic English Wordclass analysis system, TreeTagger – a language independent part-of-speech tagger, Morphy – which is a tool for German morphology and statistical part-of-speech tagging), Information Retrieval & Filtering Tools (e.g. Rubryx: Text Classification Program, seft – a Search Engine For Text, Isearch – software for indexing and searching text documents, ifile – A general mail filtering system, Bow: A Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering), Machine Learning Tools (e.g. Machine Learning Toolbox (MLT), The Machine Learning Programs Repository), FSA Tools(e.g. FSA Utilities: A Toolbox to Manipulate Finite-state Automata), HMM Tools (e.g. Hidden Markov Model (HMM) Toolbox, Discrete HMM Toolkit, A HMM mini-toolkit), Language Modeling Tools(e.g. Maximum Entropy Modeling Toolkit, Trigger Toolkit, Language modeling tools), Corpus Tools (e.g. WebCorp, Multext: i.e. Multilingual Text Tools and Corpora, TACT- i.e. Text Analysis Computing Tools, Textual Corpora and Tools for their Exploration). Some more tools include DR-LINK (Document Retrieval using LINGuistic Knowledge) system demonstrating the capabilities of NLP for Information Retrieval [Liddy *et al*, 2000], NLPWin: an NLP system from Microsoft that accepts sentences and delivers detailed syntactic analysis, together with a logical form representing an abstraction of the meaning

[Elworthy, 2000]. Waldrop (2001) has described the features of three NLP software packages, viz. Jupiter: a product of the MIT research Lab that works in the field of weather forecast, Movieline: a product of Carnegie Mellon that talks about local movie schedules, and MindNet from Microsoft Research, a system for automatically extracting a massively hyperlinked web of concepts.

2.2. Review of Research Work in Machine Learning

Machine learning is a vast field and there has been a lot of research in this area. Here we discuss the literature relevant to our thesis. Machine learning studies algorithms capable of improving their performance automatically when provided with additional knowledge regarding the specified domain. As discussed earlier, successful use of machine learning techniques depends on availability of sufficient quantities of labeled data. However, obtaining a large labeled data set becomes very expensive, particularly for the complex real-world tasks where machine learning techniques are most useful. As stated, active learning provides a way to reduce the labeling costs by labeling only the most useful instances for learning. The learning algorithm selects only those instances for annotation that are required to learn an accurate classifier [Cohn *et al.*, 1994]. Hence active learning algorithms provide much higher accuracy rates using small number of labeled examples and selecting the data from which it learns. An active learner can ask different queries in the form of unlabeled examples that are to be labeled by a human annotator. A lot of research has been carried out in this field, therefore we will describe these works along several dimensions.

2.2.1. Active Learning Scenarios

There are different circumstances in which the learner may ask queries. The learner may construct their own examples (membership query synthesis), request certain types of examples (pool-based sampling), or determine which of the unlabeled examples to query and which to discard (selective sampling). These different scenarios also determine the different sources from which the unlabeled instances are presented for annotation.

2.2.1.1. Membership Query Synthesis

In the membership query synthesis [Angluin, 1988], the learner may construct its own examples i.e. the learner may ask for labels for any unlabeled example in the input space. It also includes the queries that the learner generates anew, rather than the ones that are sampled from some underlying distribution. Query synthesis has been shown to be efficient for finite problem domains [Angluin, 2001]. It has also been extended to regression learning tasks, for example learning to predict the absolute coordinates of a robot hand [Cohn *et al.*, 1996].

In many situations query synthesis has been used efficiently however it has some disadvantages too. One of the drawbacks is that the labeling of such random instances cannot be easy if human annotator does the annotations. For example, Baum and Lang (1992) used membership query learning along with human annotators oracles for training a neural network to classify handwritten characters. They had to face an unexpected problem: most of the query images that the learner generated contained no meaningful and recognizable symbols. They only consisted of artificial characters that were meaningless. Therefore, membership query synthesis for natural language processing tasks creates meaningless streams of text or speech that are nothing more than garbage. This method usually generates meaningless examples which are hard to label as the learner is able to request a label from any possible instance from the input space and ignores the underlying sample distribution. The stream-based and pool-based scenarios have been developed to solve the above mentioned limitations. Systems using membership query syntheses have been implemented practically [King *et al.*, 2004]. In these systems an application of the membership query synthesis has been described in which a robot scientist has been shown executing a series of experiments in order to discover pathways of metabolism in yeast. In this application, a mixture of chemical solutions can be regarded as an instance and a label can be whether or not the mutant thrived in the growth medium. All experiments have been carried out autonomously using active machine learning, and physically carried out using a robot. This method reduced the experimental costs by three-fold as compared to when the least expensive experiment is run, and resulted in a 100-fold decrease in cost compared to randomly generated experiments.

2.2.1.2. Stream-based Selection/ Selective sampling

Selective sampling [Cohn *et al.*, 1994] is another active learning scenario which can be regarded as an alternative to membership query synthesis. In this scenario the instances are presented to the learner from an infinite source of unlabeled data. The learner performs the sampling of an unlabeled instance from the actual distribution as its free (or inexpensive), and then decides whether it should pay the cost of labeling it or not. This scenario is also known as stream-based or sequential active learning, because of the fact that an unlabeled instance is drawn one at a time from the data stream, and the learner has to decide whether to query or discard it. The main point on which pool-based and stream-based active learning differ is that the whole stream cannot be observed during each round of active learning, and hence limiting the protocol as the learner is able to examine each example in a stream only once during the life span of the learner and it is suitable for many applications such as speech recognition. For uniform distribution of input, this technique behaves similar to membership query learning. However, for non-uniform distribution or unknown distribution, it is certain that queries will still be meaningful, as they come from a real underlying distribution.

There are several ways by which the decision of whether to label an instance or not can be framed. One way of determining this is to evaluate the samples using some “informativeness measure” or “query strategy” and taking a random decision, so that more informative instances are more likely to be queried [Dagan and Engelson, 1995]. In another way a region of uncertainty is found [Cohn *et al.*, 1994], i.e. finding that explicit part of the instance space which is ambiguous to the learner, and then only querying the instances which fall within this region. One way of doing this is determining a minimum threshold of an informativeness measure which defines the region and query those instances whose evaluation is above this threshold. Another more principled approach is to define the region that is still unknown to the overall model class, i.e., to the set of hypotheses consistent with the current labeled training set called the version space [Mitchell, 1982]. In other words, if any two models of the same model class (but different parameter settings) agree on all the labeled data, but disagree on some unlabeled sample, then that sample lies within the region of uncertainty. The complete and explicit calculation of this region is very expensive computationally and it must be maintained after each new query. This is the reason why approximations are used in practice [Cohn *et al.*, 1994; Dasgupta *et al.*, 2008].

The stream-based scenario has been used in many practical problems, including part-of-speech tagging [Dagan and Engelson, 1995], sensor scheduling [Krishnamurthy, 2002], and learning ranking functions for information retrieval [Yu, 2005]. Fujii *et al.* (1998) employ selective sampling in active learning for word sense disambiguation, e.g., determining if the word “bank” means land alongside a river or a financial institution in a given context (only they study Japanese words in their work). The approach not only reduces annotation effort, but also limits the size of the database used in nearest-neighbor learning, which in turn expedites the classification algorithm.

2.2.1.3. Pool-based Selection

Pool-based scenario [Lewis and Gale, 1994] of active learning is based on the assumption that a small set of labeled data L and a large pool of unlabeled data U are available. During the process of active learning, an unlabeled instance is selected by the querying function Q from the unlabeled pool. The pool is assumed to be static i.e. non-changing also called closed. The querying of instances takes place according to informativeness measure in a greedy fashion. Then the annotation of the queried instance is done and the instance is then added to the set of labeled data for the purpose of training. In pool-based active learning techniques a querying function is used for scoring each instance $x \in U$ according to their informativeness. These techniques then use this score for ranking the unlabeled elements, and finally selects the highest ranked instances.

The real world problems of machine learning for which the pool-based active learning techniques have been studied include text classification [Lewis and Gale, 1994; McCallum and Nigam, 1998b; Tong and Koller, 2001; Hoi *et al.*, 2006a], information extraction [Thompson *et al.*, 1999; Settles and Craven, 2008], image classification and retrieval [Tong and Chang, 2001; Zhang and Chen, 2002], video classification and retrieval [Yan *et al.*, 2003; Hauptmann *et al.*, 2006], speech recognition [Turet *et al.*, 2005], and cancer diagnosis [Liu, 2004] to name a few. There is a difference between stream-based and pool-based active learning. In the stream based learning the data is scanned sequentially and the query decisions are made individually. In pool based learning the entire collection is evaluated and ranked before selecting the best query.

2.2.2. Querying Strategies

The main aspect of all active learning strategies is the design of an appropriate querying function, which uses the current state of the learner and properties of the available data to select unlabeled examples for annotation. The querying function evaluates the informativeness of unlabeled instances, which can either be generated de novo or sampled from a given distribution. There have been many proposed ways of designing a good querying function. Some of them are surveyed below.

2.2.2.1. Uncertainty Sampling

Uncertainty sampling [Lewis and Gale, 1994] is the simplest and most widely used query framework where the learner selects instances for which its prediction is most uncertain i.e. about which it is least confident how to label. This approach is often straightforward for probabilistic learning models. For example, when using a probabilistic model for binary classification, an uncertainty sampling strategy simply queries the instance whose posterior probability of being positive is nearest 0.5 [Lewis and Gale, 1994; Lewis and Catlett, 1994]. For many learning algorithms, a widely used method of uncertainty sampling is to select instances for which their predicted label is least confident, either from a probabilistic viewpoint or through a margin-based analogue [Lewis and Gale, 1994; Tong and Koller, 2001; Schohn and Cohn, 2000; Culotta and McCallum, 2005; Roth and Small, 2006b; Settles and Craven, 2008].

A more general uncertainty sampling strategy uses entropy [Shannon, 1948] as an uncertainty measure:

$$\Phi^{\text{ENT}}(\mathbf{x}) = -\sum P(y_i|\mathbf{x}) \log P(y_i|\mathbf{x}),$$

where Φ represents a query strategy, which is a function used to evaluate the informativeness of a query, \mathbf{x} represents the best query instance which maximizes this function, and y_i ranges over all possible labeling. The entropy-based approach can be generalized easily to probabilistic multi-label classifiers and probabilistic models for more complex structured instances, such as sequences [Settles and Craven, 2008] and trees [Hwa, 2004]. An alternative to entropy in these more complex settings involves querying the instance whose best labeling is the least confident:

$$\Phi^{\text{LC}}(\mathbf{x}) = 1 - P(y^*|\mathbf{x}),$$

where $y^* = \operatorname{argmax} P(y|\mathbf{x})$ is the most likely class labeling. This sort of strategy has been shown to work well, for example, with conditional random fields or CRFs [Lafferty *et al.*, 2001] for

active learning in information extraction tasks [Culotta and McCallum, 2005; Settles and Craven, 2008]. Uncertainty sampling strategies may also be employed with non-probabilistic models. One of the first works to explore uncertainty sampling used a decision tree classifier [Lewis and Catlett, 1994] by modifying it to have probabilistic output. Similar approaches have been applied to active learning with nearest-neighbor (“memory-based” or “instance-based”) classifiers [Fujii *et al.*, 1998; Lindenbaum *et al.*, 2004], by allowing each neighbor to vote on the class label of x , with the proportion of these votes representing the posterior label probability. Tong and Koller (2000) also experiment with an uncertainty sampling strategy for support vector machines, or SVMs [Cortes and Vapnik, 1995], that involves querying the instance closest to the linear decision boundary. This last approach is analogous to uncertainty sampling with a probabilistic binary linear classifier, such as logistic regression or naive Bayes [Kosmopoulos *et al.*, 2008].

2.2.2.2. Query-By-Committee

The query-by-committee (QBC) framework [Seung *et al.*, 1992; Freund *et al.*, 1997; Fine *et al.*, 2002] is similar to uncertainty sampling, but is distinguished by using an ensemble of experts to select instances for annotation. In QBC, a committee of learned models is trained using the labeled data and a querying function is derived through a voting mechanism. The QBC approach involves maintaining a committee C of models which are all trained on the current labeled set L , but represent competing hypotheses. Each committee member is then allowed to vote on the labelings of query candidates. The most informative query is considered to be the instance about which they most disagree. The basic principle of QBC approach is to minimize the version space. Version space is the set of hypotheses that are consistent with the current labeled training data L . If machine learning is considered as the search for the best model within the version space, then the aim of active learning is to limit the size of this space as much as possible with as few labeled instances as possible in order to make the search more precise. QBC does exactly this by querying in controversial regions of the version space.

Two things are necessary in a QBC framework, one is to construct a committee of models that approximate different regions of the version space and the other is to have some measure of disagreement among them. Seung *et al.* (1992) accomplish the first task simply by sampling a committee of two random hypotheses that are consistent with L . For generative model classes, this can be done more generally by randomly sampling models from some posterior distribution

$P(\theta|L)$. For example, McCallum and Nigam (1998b) do this for naive Bayes by using the Dirichlet distribution over model parameters, whereas Dagan and Engelson (1995) sample HMMs by using the Normal distribution. For other model classes, such as discriminative or non-probabilistic models, Abe and Mamitsuka (1998) have proposed query-by-boosting and query-by-bagging, which employ the well-known ensemble learning methods boosting [Freund and Schapire, 1997] and bagging [Breiman, 1996] to construct committees. Melville and Mooney (2004) propose another ensemble-based method which encourages diversity among committee members. For measuring the degree of disagreement, two main approaches have been proposed: vote entropy [Dagan and Engelson, 1995] and average KL-divergence [McCallum and Nigam, 1998b]. There is no consensus on the appropriate committee size to use, which may in fact vary by model class or application. However, even small committee sizes (e.g., two or three) have been shown to work well in practice [Seung *et al.*, 1992; McCallum and Nigam, 1998b; Settles and Craven, 2008]. Aside from the QBC framework, several other query strategies attempt to minimize the version space as well. For example, Cohn *et al.* (1994) describe a related selective sampling algorithm for neural networks using a combination of the “most specific” and “most general” models, which lie at two extremes the version space given the current labeled examples in the training set L . Tong and Koller (2000) propose a pool-based query strategy that tries to minimize the version space for support vector machine classifiers directly. The membership query algorithms of Angluin (1988) and King *et al.* (2004) can also be interpreted as synthesizing de novo instances that limit the size of the version space. However, Haussler (1994) shows that the size of the version space can grow exponentially with the size of L . This means that, in general, the version space of an arbitrary model class cannot be explicitly represented in practice. The QBC framework, rather, uses a committee which is a subset-approximation of the full version space.

2.2.2.3. Unreliability Sampling

Another recently developed strategy for designing a querying function is unreliability sampling [Becker, 2008]. The basic premise of this framework is that instances should be selected which have parameters which have not observed sufficient data for confident estimation. An early instantiation of this method was active learning for syntactic parsing, where unlabeled instances which cause the current parsing model to fail are used to request labels from the expert

[Thompson *et al.*, 1999]. Following the same basic principles, this paradigm has been extended for improvements in active learning for syntactic parsing [Becker and Osborne, 2005] and active learning for machine translation [Haffari *et al.*, 2009]. Recent work on confidence-weighted active learning [Dredze and Crammer, 2008] applies a similar philosophy by selecting examples with parameters possessing high variance during estimation. As opposed to uncertainty sampling, which selects examples for which the prediction has low confidence, unreliability sampling selects those instance for which an accurate measure of certainty cannot be computed.

2.2.2.4. Expected Model Change

A much more recently formalized approach for designing a querying function is to select instances which exhibit the greatest expected model change [Settles and Craven, 2008] i.e. that would impart the greatest change to the current model if we knew its label. As opposed to selecting instances for which the learner is least confident, the expected model change selects instance for which there is an expectation of significant change in between the current hypothesis and the resulting induced hypothesis if the instance was labeled. This strategy was noted earlier in the context of selecting instances for learning an SVM [Bordes *et al.*, 2005], but without an accurate estimate of model change, they relied on a margin-based uncertainty method. The intuition behind this framework is that those instances will be preferred that are likely to most influence the model (i.e., have greatest impact on its parameters), regardless of the resulting query label. This approach has been shown to work well in empirical studies, but can be computationally expensive if both the feature space and set of labelings are very large.

2.2.2.5. Estimated Error Reduction

A traditionally less popular strategy gaining increasing attention is the use of querying functions which attempt to directly minimize the generalization error. Under this framework, each instance is scored with respect to the expected reduction in future error if labeled and added to the training data. This method is theoretically appealing as it attempts to directly minimize error, the true task at hand. Although shown to be empirically effective, the drawback to querying by expected error reduction is the computation required to estimate expected error and compute an updated model for each possible labeling for each unlabeled instance. However, this approach has been shown very successful when methods such as sub sampling the unlabeled pool with a

naive Bayes classifier [Roy and McCallum, 2001], exact incremental updates with Gaussian random fields [Zhu *et al.*, 2003], and approximate training methods with logistic regression [Guo and Greiner, 2007].

Unfortunately, estimated error reduction may also be the most prohibitively expensive query selection framework. Not only does it require estimating the expected future error over U for each query, but a new model must be incrementally re-trained for each possible query labeling, which in turn iterates over the entire pool. This leads to a dramatic increase in computational cost. For some model classes such as Gaussian random fields [Zhu *et al.*, 2003], the incremental training procedure is efficient and exact thus making this approach fairly practical. For a many other model classes, this is not the case.

A statistically well motivated querying function strategy is selecting instances which minimize variance [Cohn *et al.*, 1996]. Given the observation that expected generalization error can be decomposed into bias and variance components [Geman *et al.*, 1992], the variance minimization strategy is to select instances for which once labeled and added to the training data will result in the greatest reduction in variance and thus generalization error. As this approach is only feasible for definitions of variance which are smooth and differentiable, it has only been applied to problems such as regression and neural networks [Cohn *et al.*, 1996]. Related and more appropriate for the standard active learning settings is selection based upon the Fischer information associated with a prediction [Zhang and Oles, 2000; Hoi *et al.*, 2006; Settles and Craven, 2008], which also require approximation techniques to calculate efficiently.

2.2.2.6. Density-Weighting Methods

One unfortunate property of many active learning querying functions is that they are relatively noise intolerant, motivating the study of techniques which weigh instances by how representative they are of the input distribution of the data, referred to as density-weighted querying functions. Pre-clustering the data and selecting examples which represent each cluster has been demonstrated a very successful for querying representative instances [Nguyen and Smeulders, 2004; Donmez *et al.*, 2007; Xu *et al.*, 2007]. These methods are particularly beneficial when learning from only a few instances, which is done early in the active learning process. Density-weighting formulations have also been studied for query-by-committee [McCallum and Nigam, 1998b] and in the context of sequence prediction [Settles and Craven, 2008]. The main idea is

that informative instances should not only be those which are uncertain, but also those which are “representative” of the input distribution (i.e., inhabit dense regions of the input space). Fujii *et al.* (1998) explored a query strategy for nearest-neighbor methods that selects queries that are unlike the labeled instances already in L , and most similar to the unlabeled instances in U .

2.2.3. Structured Outputs

Several important learning problems involve predicting structured outputs on instances, such as sequences and trees. In these problems multiple local predictions must be combined to form a coherent structure. These models have garnered significant interest in the NLP and other application communities as they can effectively incorporate information from multiple sources regarding many interdependent prediction tasks. As structured output labels are generally more expensive to obtain, there has been a corresponding interest in reducing labeling requirements in these settings. In the context of active learning, there has been some recent work regarding learning in structured output spaces including work on active learning for HMMs [Dagan and Engelson, 1995; Scheffer and Wrobel, 2001; Anderson and Moore, 2005], CRFs [Culotta and McCallum, 2005; Settles and Craven, 2008] and structured Perceptron [Roth and Small, 2006b]. More application targeted includes active learning for probabilistic context free grammars (PCFGs) [Baldrige and Osborne, 2004; Hwa, 2004]. Also, closely related works for settings more complex than binary classification include active learning for multiclass classification [Yan *et al.*, 2003; Brinker, 2004] and active learning for ranking data [Brinker, 2004; Donmez and Carbonell, 2008].

Active learning, most notably pool-based selection, has been applied to many NLP applications including text/spam classification [Lewis and Gale, 1994; Liere and Tadepalli, 1997; McCallum and Nigam, 1998a; Schohn and Cohn, 2000; Tong and Koller, 2001; Hoi *et al.*, 2006a; Schein and Ungar, 2007; Dredze and Crammer, 2008; Zhu *et al.*, 2008a], chunking [Ngai and Yarowsky, 2000], part of speech tagging [Dagan and Engelson, 1995], named entity recognition [Scheffer and Wrobel, 2001; Shen *et al.*, 2004; Becker *et al.*, 2005; Jones, 2005; Kim *et al.*, 2006; Vlachos, 2006; Tomanek *et al.*, 2007; Laws and Schutze, 2008], information extraction [Thompson *et al.*, 1999; Scheffer *et al.*, 2001; Finn and Kushmerick, 2003; Jones *et al.*, 2003; Culotta and McCallum, 2005; Culotta *et al.*, 2006; Roth and Small, 2008; Settles and Craven, 2008], prepositional phrase attachment [Hwa, 2004; Becker, 2008], syntactic parsing [Thompson *et al.*,

1999; Tang *et al.*, 2002; Hwa, 2004; Becker and Osborne, 2005], word sense disambiguation [Chen *et al.*, 2006; Chan and Ng 2007; Zhu and Hovy, 2007], semantic role labeling (Roth and Small, 2006b) and machine translation [Haffari *et al.*, 2009; Haffari and Sarkar, 2009].

A framework and objective functions have been introduced for active learning in three fundamental HMM problems: model learning, state estimation, and path estimation. In addition, a new set of algorithms has been described for efficiently finding optimal greedy queries using these objective functions. The algorithms are fast, i.e., linear in the number of time steps to select the optimal query and we present empirical results showing that these algorithms can significantly reduce the need for labelled training data [Anderson and Moore, 2005].

Many classification problems with structured outputs can be regarded as a set of interrelated sub-problems where constraints dictate valid variable assignments. The standard approaches to these problems include either independent learning of individual classifiers for each of the sub-problems or joint learning of the entire set of classifiers with the constraints enforced during learning. An intermediate approach has been proposed where these classifiers are learnt in a sequence using previously learned classifiers to guide learning of the next classifier by enforcing constraints between their outputs. A theoretical motivation has been provided to explain why this learning protocol is expected to outperform both alternatives when individual problems have different 'complexity'. This analysis motivates an algorithm for choosing a preferred order of classifier learning. This technique has been evaluated on artificial experiments and on the entity and relation identification problem where the proposed method outperforms both joint and independent learning. [Bunescu, 2008].

The success of interactive machine learning systems depends both on the machine and on the human performance. An understanding of machine capabilities and limitations should inform interaction design, while the abilities, preferences, and limitations of human operators should inform the choice of inputs, outputs, and performance requirements of machine learning algorithms. A relevant example from the past work is Arnauld system [Krzysztof and Daniel, 2005] for active preference elicitation. A lot of previous work in that area solicited user feedback in the form numerical ratings over possible outcomes. However, unless the rating scale is well grounded, people tend to be inconsistent and unreliable providing this type of feedback. What works much more robustly is pairwise comparison queries, where the person only has to state which of two possible outcomes he or she prefers [Krzysztof and Daniel, 2005]. Adopting this

input interaction, however, requires developing a new learning algorithm. In turn, to account for the limitations of the algorithm, the example critiquing interaction [Pearl and Chen, 2009] has been implemented to allow people to manually direct the learning once the active learning process no longer resulted in rapid improvements in the model quality. Work has been done on incorporating richer user feedback into interactive machine learning systems. Typically, machine learning algorithms only solicit labels from the users but several projects e.g. [Gregory *et al.*, 2007] have shown that incorporating richer feedback-that captures the user's rationale-leads to faster and more generalizable learning. So far, this feedback has been limited to feature relevance. Is this the best or the only type of rich feedback that can be elicited from users? A preliminary study has been conducted in the context of preference elicitation for an e-commerce application to understand what types of feedback people naturally provide, and what the value of these different types of feedback might have for the speed and quality of learning. Specifically, users were asked to answer a set of pair wise comparison questions regarding digital cameras and their choices has been recorded as well as free form explanations of their choices.

End-user interactive concept learning is a technique for interacting with large unstructured datasets, requiring insights from both human-computer interaction and machine learning. This note re-examines an assumption implicit in prior interactive machine learning research i.e. interaction should focus on the question "*what class is this object?*". Amershi, S. *et al* (2010) have broadened interaction to include examination of multiple potential models while training a machine learning system. They evaluated this approach and found that people naturally adopted revision in the interactive machine learning process and that this improved the quality of their resulting models for difficult concepts.

M. Kristan *et al* (2009) have proposed a Gaussian-kernel-based online kernel density estimation which can be used for applications of online probability density estimation and online learning. This approach generates a Gaussian mixture model of the observed data and allows online adaptation from positive examples as well as from the negative examples. The adaptation from the negative examples is realized by a novel concept of unlearning in mixture models. Low complexity of the mixtures is maintained through a novel compression algorithm. In contrast to other approaches, this approach does not require fine-tuning parameters for a specific application, they have not assumed specific forms of the target distributions and temporal constraints have not been assumed on the observed data. The strength of the proposed approach

has been demonstrated with examples of online estimation of complex distributions, an example of unlearning, and with an interactive learning of basic visual concepts.

Very recently there has been work on actively selecting examples with the intention of labeling properties regarding features. The earliest example of this work is the tandem learning algorithm where the expert iteratively queries the expert for instance labels and then feature labels. This idea of labeling both instances and features simultaneously has been further pursued in the active dual supervision model [Sindhwani *et al.*, 2009]. Even more recently, the generalized expectation criteria has been incorporated into the active learning framework to present instances to the domain expert for the explicit purpose of incorporating domain knowledge by labeling features [Druck *et al.*, 2009]. The learning from measurements model [Liang *et al.*, 2009] also works along this vein by deriving a framework based on Bayesian experimental design to select instances for which the largest expected information gain will be achieved if the feature is labeled.

In most of the active learning research, queries are selected in serial, i.e., one at a time. However, sometimes the training time required to induce a model is slow or expensive, as with large ensemble methods and many structured prediction tasks. Consider also that sometimes a distributed, parallel labeling environment may be available, e.g., multiple annotators working on different machines at the same time on a network. In both of these cases, selecting queries in serial may be inefficient. By contrast, batch-mode active learning allows the learner to query instances in groups, which is better suited to parallel labeling environments or models with slow training procedures.

Myopically querying the “N-best” queries according to a given instance-level query strategy often does not work well, since it fails to consider the overlap in information content among the “best” instances. To address this, a few batch-mode active learning algorithms have been proposed. Brinker (2003) considers an approach for SVMs that explicitly incorporates diversity among instances in the batch. Xu *et al.* (2007) propose a similar approach for SVM active learning, which also incorporates a density measure. Specifically, they query cluster centroids for instances that lie close to the decision boundary. Hoi *et al.* (2006a,b) extend the Fisher information framework to the batch-mode setting for binary logistic regression. Most of these approaches use greedy heuristics to ensure that instances in the batch are both diverse and informative, although Hoi *et al.* (2006b) exploit the properties of submodular functions to find

near-optimal batches. Alternatively, Guo and Schuurmans (2008) treat batch construction for logistic regression as a discriminative optimization problem, and attempt to construct the most informative batch directly. For the most part, these approaches show improvements over random batch sampling, which in turn is generally better than simple “N-best” batch construction.

In some learning problems, the cost of acquiring labeled data can vary from one instance to the next. If our goal in active learning is to minimize the overall cost of training an accurate model, then reducing the number of labeled instances does not necessarily guarantee a reduction in overall labeling cost. One proposed approach for reducing annotation effort in active learning involves using the current trained model to assist in the labeling of query instances by pre-labeling them in structured learning tasks like parsing [Baldrige and Osborne, 2004] or information extraction [Culotta and McCallum, 2005]. However, such methods do not actually represent or reason about labeling costs. Instead, they attempt to reduce cost indirectly by minimizing the number of annotation actions required for a query that has already been selected.

Another group of cost-sensitive active learning approaches explicitly accounts for varying label costs in active learning. Kapoor *et al.* (2007) propose one approach that takes into account both labeling costs and estimated misclassification costs. In this setting, each candidate query is evaluated by summing the labeling cost for the instance and the expected future misclassification costs that would be incurred if the instance were added to the training set. Instead of using real costs, however, their experiments make the simplifying assumption that the cost of labeling a voice mail message is a linear function of its length (e.g., ten cents per second). King *et al.* (2004) use a similar active learning approach in an attempt to reduce actual labeling costs. They describe a “robot scientist” which can execute a series of autonomous biological experiments to discover metabolic pathways, with the objective of minimizing the cost of materials used (i.e., the cost of an experiment plus the expected total cost of future experiments until the correct hypothesis is found).

As previously stated, the primary research issue for active learning is the design of an appropriate querying function. However, it is possible that different querying functions work better for different regions of the active learning cycle. For example, a querying function using density-weighted selection is very helpful for initial queries, but uncertainty sampling is more effective once the classifier is relatively stable [Donmez *et al.*, 2007]. Baram *et al.* (2004) examine scenarios where several querying functions are employed by being cast in the multi-

armed bandit framework, where querying functions are selected which explicitly follow an exploration and exploitation cycles. In addition to selecting appropriate querying functions for different operating regions, as the overall goal of active learning is to reduce total annotation, it is also useful to know when maximal performance is achieved such that unnecessary actions will be avoided, referred to as a stopping criterion [Schohn and Cohn, 2000; Campbell *et al.*, 2000; Tomanek *et al.*, 2007; Vlachos, 2008; Dimitrakakis and Savu-Krohn, 2008; Laws and Schutze, 2008; Zhu *et al.*, 2008a,b]. The critical aspect of deriving a stopping criterion is a method for autonomously determining the performance of the current learner hypothesis (i.e. without development or testing data). Other works have used a self-estimated measure of active learning performance to determine different operating regions which require different querying functions to be most effective [Baram *et al.*, 2004; Donmez *et al.*, 2007; Roth and Small, 2008].

Chapter 3
Background

This chapter discusses concepts that are relevant to the work presented in this thesis. The sections that follow discuss basic concepts about supervised machine learning and active learning. Section 3.1 discusses basics of supervised learning as well as the terminology and the procedure used in supervised learning algorithms. It provides an idea about version space and feature space and explains two important examples of supervised learning: classification and regression. Section 3.2 discusses machine learning for complex problems i.e. learning structured instances and learning pipeline models. Section 3.3 discusses pool-based active learning.

3.1. Supervised Learning

Supervised learning [Kotsiantis, 2007] is the machine learning task in which the algorithms reason from externally supplied instances to produce general hypothesis, which then make predictions about future instances. It is the task of deriving a function from labeled training data. In the supervised machine learning problem a function maps the inputs to the desired outputs by determining to which class among a set of classes a new input belongs to. This is done with the help of the training data which consists of the instances with labelled output i.e. known class. The training data is a collection of training examples. The training examples are in the form of pairs that consist of input x and a desired output value y . The job of supervised learning algorithms is analyzing the training data and producing a function. This function can take two forms i.e. it can be a classifier if the output is discrete or it can be called as a regression function in case the output is continuous. The system is provided with labelled instances represented as (x, y) and the objective of supervised learning systems is to determine the label y for each new input x that it sees in future. When y is a real number, the task is called regression, when it is a set of discrete values, the task is called classification. For any valid input, the derived function should be able to predict the correct output value. In order to be able to predict the correct output, the learning algorithm should have to generalize from the labelled training data to unseen situations in a reasonable way. Supervised learning is the learning based on training data. The datasets used by machine learning algorithms consists of a number of instances that are represented using the same set of features. In supervised learning the instances are given with known labels (the corresponding correct outputs) in contrast to unsupervised learning, where instances are unlabeled.

As stated earlier, in supervised machine learning a function maps the inputs to the desired outputs by determining which of a set of classes a new input belongs to. The mapping function can be represented by f . h denotes the hypothesis about the function to be learned. Inputs are represented as $X = (x_1, x_2, \dots, x_n)$ and outputs as $Y = (y_1, y_2, \dots, y_n)$ [Nilsson, 2005]. Therefore, hypothesis or the prediction function can be written as

$$h : X \rightarrow Y$$

h is the function of vector-valued input and is selected on the basis of training set of m input vector examples i.e.

$$X = (x_1, x_2, \dots, x_n) \rightarrow \boxed{h} \rightarrow h(X)$$

Training set = $\{ X_1, X_2, \dots, X_m \}$

Therefore, the predicted value can be given as

$$y = h(x) = \operatorname{argmax}_{y' \in Y} f(x, y')$$

3.1.1. Terminology

The variables used in supervised machine learning are:

- x_1, x_2 , and so on represent the input values, and X represents the input domain, such that $x \in X$.
- y_1, y_2 , and so on represent the output values, and Y represents the output space, such that $y \in Y$.
- There are a number of different types of machine learning problems which can be defined by the output space i.e., binary classification in which case $Y = \{-1, 1\}$, regression in which case $Y = \mathbb{R}$, multiclass classification in which case $Y = \{w_1, w_2, \dots, w_k\}$.
- The probability distribution from which the supervised data is drawn is represented by $D_{X \times Y}$

- Φ represents the feature vector generating procedure. Input to this function is the members of the input space X and returns a d -dimensional feature vector $\mathbf{x} \in \mathbb{R}^d$. This vector is then used as the input by the learning algorithm.

$$\Phi : X \rightarrow \Phi(X)$$

where, $\Phi(X)$ represents the input domain after Φ is applied to all the members $x \in X$.

- H represents the hypothesis space used by a machine learning system which is defined as the set of all possible hypotheses that the machine learning system can return. It is denoted as

$$H : \Phi(X) \rightarrow Y$$

and the learned hypothesis h is selected from H ,

$$h \in H$$

- L represents the loss function which can be defined as a function which measures the difference between estimated and the true values for some data element and in case of machine learning it can be defined as the measure of divergence between two output elements. The frequently used loss function in learning problems is the 0-1(zero-one) loss function $L(y', y) = 1$ if y' is not equal to y and 0 otherwise.
- S represents the training sample drawn from the probability distribution $D_{\Phi(X)*Y}$

$$S = \{(\mathbf{x}_i, y_i)\} \text{ where } i = 1 \text{ to } m.$$

After defining all the variables, we can now easily provide a proper definition of a machine learning algorithm or a learner. A machine learning algorithm can be defined as an algorithm which when provided with a hypothesis space H , a loss function L , and a training set S of m training examples drawn from a probability distribution $D_{\Phi(X)*Y}$, returns a hypothesis function $\hat{h} \in H$ that minimizes the expected loss L on a randomly drawn example from $D_{\Phi(X)*Y}$,

$$\hat{h} = \operatorname{argmin}_{h' \in H} E_{(x,y) \sim D_{\Phi(X)*Y}}(L(h'(\mathbf{x}), y)).$$

In theoretical terms, we would wish to design the above mentioned algorithm however in practical situations it becomes infeasible to develop such algorithms. In practical situations the

algorithms actually minimize the empirical loss since only a finite set of training examples are given and $D_{\Phi(X)*Y}$ is unknown. In such cases the learning algorithm returns the hypothesis h as,

$$\hat{h} = \operatorname{argmin}_{h' \in H} \sum L(h'(\mathbf{x}_i), y) \text{ where } i = 1 \text{ to } m$$

Zero-one loss function $L_{0/1}$ forms the basis of classification therefore minimizing this function makes much sense however it becomes intractable for the linear classifiers. Therefore, instead of minimizing the ideal loss function a number of learning algorithms minimize a differentiable function as a substitute for the ideal loss function. Margin-based algorithms [Allwein *et al.*, 2000; Pelosof *et al.*, 2010] are an example of such algorithms. The terms used in such learning algorithms are discussed as under:

- F represents a set of hypothesis scoring functions i.e.

$$F : \Phi(X) * Y \rightarrow \mathbb{R} \text{ such that } \hat{y} = h(\mathbf{x}) = \operatorname{argmax}_{y' \in Y} f_{y'}(\mathbf{x}).$$

- ρ represents the margin of an instance. It is a non-negative real-valued function which is equal to 0 if and only if $\hat{y} = y$ and its magnitude is related to the confidence of a prediction \hat{y} for the given input \mathbf{x} relative to a specific hypothesis h .

$$\rho : \Phi(X) * Y * F \rightarrow \mathbb{R}^+$$

- $L : \rho \rightarrow \mathbb{R}^+$ represents the margin-based loss function which measures the difference between the predicted output and the true output based upon its margin relative to a specified hypothesis.

Thus the margin-based algorithms return a hypothesis scoring function $\hat{f} \in F$ which minimize the empirical loss over the training examples to select a hypothesis scoring function

$$\hat{f} = \operatorname{argmin}_{f \in F} \sum L(\rho(\mathbf{x}, y, f))$$

3.1.2. Version Space and Feature Space

In this section we provide some idea about the version space and the feature space. A version space [Mitchell, 1977; Herbrich *et al.*, 2004] can be defined as the set of hypotheses within a given hypothesis space H that are consistent with the observed training examples. It can also be defined as the subset of all hypotheses which can label every instance from a given sample correctly. Version space provides an important framework for active learning.

Version space can be represented by two sets of hypotheses. The first one is called most specific consistent hypotheses, and the other one is called the most general consistent hypotheses. In both these types the term "consistent" means that the hypotheses are consistent with the observed data. The most specific hypotheses include all the positive training instances, and as small area of the remaining feature space as possible. If these hypotheses are further reduced, then a positive training instance will be excluded and the hypotheses will become inconsistent. The most general hypotheses include the positive instances and as much of the remaining feature space as possible without including any negative instance. If these hypotheses are enlarged any further, then a negative instance will be included making the hypotheses inconsistent. Figure 3.1 [Dubois *et al.*, 2002] shows the two hypotheses sets in version space. GB stands for general boundary and SB stands for specific boundary.

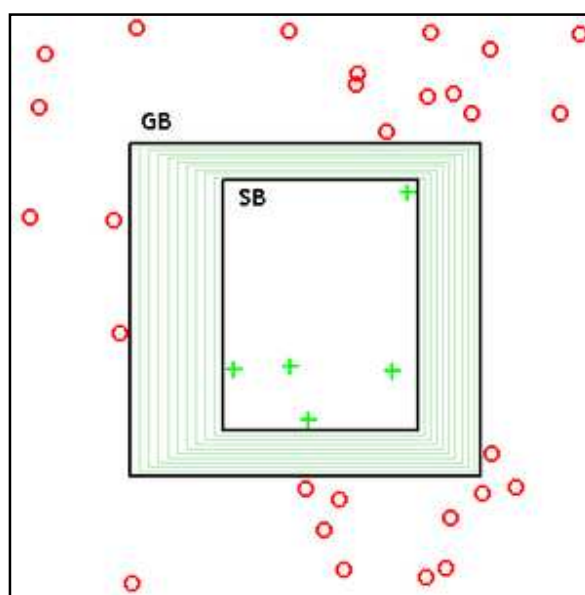


Figure 3.1: Version Space

Further we can call a hypothesis h as being consistent with a training sample S if and only if $h(\mathbf{x}) = y$ for each $(\mathbf{x}, y) \in S$. Also, if we have a hypothesis space H and a training sample S then the version space V with respect to H is the set of all hypotheses $h \in H$ which are consistent with S .

As stated earlier in this chapter, Φ represents the feature vector generating procedure. Input to this function is the members of the input space X and returns a d -dimensional feature vector $\mathbf{x} \in \mathbb{R}^d$, i.e.

$$\Phi(x) \rightarrow \mathbf{x}$$

In machine learning, a feature can be defined as a measurable property of an item or a phenomenon under observation and a feature vector can be defined as an n -dimensional vector of numerical features representing some item or the set of features of a given data instance. Machine learning problems require a lot of processing and statistical analysis. Therefore in order to facilitate such analysis machine learning algorithms need numerical features or numerical representation of items. For example, in case of representing an image, the feature values correspond to the pixels and in case of text they correspond to the term occurrence frequencies. Thus we can define feature space as the space associated with these feature vectors.

3.1.3. Supervised Machine Learning Procedure

For solving any problem the supervised machine learning algorithm follows a number of steps. This section discusses each of these steps.

- The first and foremost step is the *collection of the data* required for solving a particular problem. It consists of identifying all the important features or attributes that are most relevant to the problem under study.
- The second step is the *pre-processing* [Zhang *et al.*, 2002] of data. The data collected in the first step is not directly suitable for training and therefore requires some processing before it can be used for example it may have missing feature values or noise. A number of pre-processing methods have been developed and the decision of deciding which one to use varies according to the situations. If the collected data contains some missing features then a method for handling missing data [Batista & Monard, 2003] is used. Similarly, there are methods for detecting and handling noise [Hodge & Austin, 2004].
- The third step is *feature subset selection*. It consists of recognizing and eliminating the features that are redundant or that are not relevant for the problem under study [Yu & Liu, 2004]. It increases the efficiency of the learning algorithms by decreasing the

dimensionality of the data. In order to develop more accurate and efficient classifiers a process called feature construction is used. In this process new features are constructed from the existing basic features [Markovitch & Rosenstein, 2002] in situations where many features depend on one another.

- The fourth step is *evaluating the accuracy of the classifier*. This step decided whether the classifier is fit to be used or some modifications are required. The evaluation of the classifier depends on the prediction accuracy (Number of correct predictions / Total number of predictions). The classifier's accuracy can be estimated in three ways:
 - i. First one is the splitting of the training set and using two-thirds for training and the other third for estimating performance.
 - ii. Second one is called cross-validation. In this technique mutually exclusive and same-sized subsets are created by dividing the training set. For each subset the classifier is trained on the union of all the other subsets. Using this technique the error rate of the classifier is calculated by the average of the error rate of each subset.
 - iii. Third one is called leave-one-out validation. It is a type of cross validation in which all the test sets contain single instance.

If the error rate evaluation shows that the classifier is not efficient enough or is unacceptable then the algorithm returns to previous stage and some factors are examined again for example features are checked again to eliminate irrelevant features, or the size of training set is checked again. Some other problems that might occur include too high dimensionality of the problem or imbalanced dataset [Japkowicz & Stephen, 2002]. However, if the evaluation shows satisfactory results then the classifier is available for use.

3.1.4. Examples of Supervised Machine Learning: Classification and Regression

Among many other learning examples, classification and regression are two important supervised learning problems. This section discusses each of these techniques with examples. As discussed earlier, the training data in supervised learning is a collection of training examples. The training examples are in the form of pairs that consist of input x and a desired output value

y. The job of supervised learning algorithms is analyzing the training data and producing a function. This function can take two forms i.e. it can be a classifier if the output is discrete or it can be called as a regression function in case the output is continuous. The system is provided with labelled instances represented as (x, y) and the objective of supervised learning systems is to determine the label y for each new input x that it sees in future. When y is a real number, the task is called *regression*, when it is a set of discrete values, the task is called *classification*.

Classification

In machine learning, we can define classification [Michie *et al.*, 1994] as the task of determining to which class among a set of classes a new input belongs. This is done with the help of the training data which contains the instances whose class is known. In classification, there are a number of classes and the goal is to develop a rule that classifies a new input into one of the existing classes. Classification is an example of supervised learning and its corresponding unsupervised method is called clustering in which there are a set of observations and the goal is to establish the existence of clusters or classes in the data i.e. the data is grouped into categories based on some measure of similarity. The algorithm that is used for classification is called a classifier. The word "classifier" can be also used to represent the function implemented by a classification algorithm that maps input data to a given class. There are certain issues which must be taken care of while developing a classifier such as accuracy, speed, comprehensibility, and time to learn a classification rule.

Classification can be either binary classification or multiclass classification. Binary classification consists of only two classes. In multiclass classification an object can be assigned to any one of a number of classes. An example of binary classification is the classification of customers in the bank loan application. In this example, the input to the classifier is the information about the customer and the goal of the classifier is to assign the input to one of the two classes i.e. low-risk and high-risk customers. The information about the customer may include his income, savings, age, profession, past financial history and so on. In this example, a classification rule learned is of *if-then* type i.e., if the customer income is greater than some particular amount and his savings are greater than some particular amount then the customer can be classified into low-risk class else the customer will be classified into high-risk class. Such an example is called a discriminant

function which separates the examples of different classes. This function involves prediction i.e. when a rule fits the past data then correct predictions can be made for new examples. In some cases, instead of making a 0/1 (low-risk/high-risk) type decision, we may want to calculate a probability, namely, $P(Y|X)$, where X are the customer attributes and Y is 0 or 1 respectively for low-risk and high-risk. From this perspective, we can see classification as learning an association from X to Y . Then for a given $X = x$, if we have $P(Y = 1|X = x) = 0.8$, we say that the customer has an 80 percent probability of being high-risk, or equivalently a 20 percent probability of being low-risk. We then decide whether to accept or refuse the loan depending on the possible gain and loss.

There are a number of classification algorithms that have been developed. These include Fisher's linear discriminant, Logistic regression, Naive Bayes classifier, Perceptron, Support vector machines, Least squares support vector machines, k-nearest neighbour, Decision trees, Random forests, Neural networks, Bayesian networks, and Hidden Markov models.

Regression

Regression can be defined as a technique that is used for calculating the relationships between variables i.e. the relationship between a dependent variable and one or more independent variables. In other words we can say that the process of regression depicts the changes in the values of a dependent variable by varying the value of one of the independent variables while the other independent variables are kept fixed. In machine learning, regression can be defined as a technique that is used to fit an equation to a dataset. The simplest type of regression technique is linear regression. In this form of regression the formula of straight line is used i.e. $y = mx + b$ and the suitable values for m and b are estimated in order to predict the value of y on the basis of a given value of x . Another form of regression is called multiple regression. In this technique more than one input variable is used that fits more complex models, such as a quadratic equation. Applications of regression are prediction and forecasting. There are a number of techniques for using regression. Least squares regression and linear regression are parametric methods. It means the function is described in terms of a finite number of unknown parameters that are estimated from the data. Another form of regression is nonparametric regression in which the regression function is allowed to lie in a specified set of functions, which may be infinite-dimensional. In

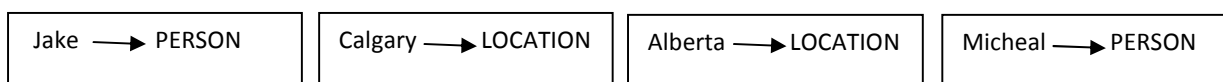
order to explain the regression technique we can take the example of a system that should be able to predict the price of a car. Inputs to the system are the car attributes such as engine capacity, mileage, brand and so on which show the worth of the car. The output is the price of the car. Such problems where the output is a number are *regression* problems. Let X denote the car attributes and Y be the price of the car. Again surveying the past transactions, we can collect a training data and the machine learning program fits a function to this data to learn Y as a function of X . The function is of the form $y = wx + w_0$ for suitable values of w and w_0 .

Regression and classification are both problems of supervised learning. In these problems, there is an input X and an output Y and the goal is to learn a mapping from input to the output. Machine learning uses an approach that assumes a model defined up to a set of parameters, i.e. $y = g(x|\theta)$ where $g(\cdot)$ is the model and θ are its parameters. Y is a number in regression and is a class code (e.g., 0/1) in the case of classification. $g(\cdot)$ is the regression function or in classification, it is the discriminant function separating the instances of different classes. The machine learning program optimizes the parameters, θ , such that the approximation error is minimized, that is, our estimates are as close as possible to the correct values given in the training set.

3.2. Machine Learning for Complex Problems

In the beginning of this chapter in Section 3.1, we have described the general framework of supervised machine learning. However, in practical environments when we want to apply machine learning to various complex problems like information extraction, a single function cannot be used to carry out the task efficiently. For example, in case of relation extraction, it is not possible for a single function to accurately identify all of the named entities and relations within a sentence. Consider the sentence given in Figure 3.2 in which we need to extract all the entities and label the relations between the entities.

Jake works in Calgary, Alberta with his brother Micheal.



Entity detection

{Jake, Calgary} \rightarrow works_in

{Jake, Micheal} \rightarrow brother_of

{Calgary, Alberta} \rightarrow located_in

{Jake, Alberta} \rightarrow works_in

Relation detection

Figure 3.2: Entity and Relation detection from text

In such cases, a more practical approach is to learn a complex model which divides the learning problem into a number of sub problems and then reassembles them to return a predicted global annotation.

3.2.1. Learning Structured Instances

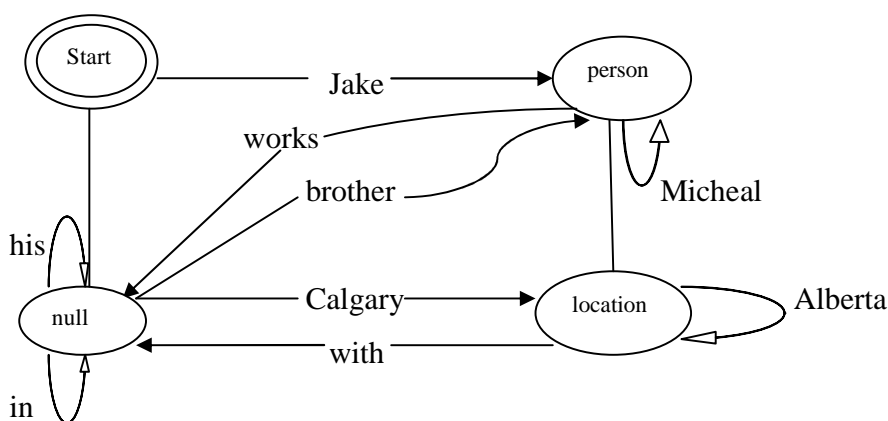
One of the important methods for solving complex problems is learning in structured output spaces. In this method, a number of local learners trained which then return a predicted global structure. Examples of such a classifier include structured support vector machines [Tsochantaridis *et al.*, 2004], hidden markov model [Rabiner, 1989], that illustrates a generative model for learning sequential structures, conditional random fields [Lafferty *et al.*, 2001], structured perceptron [Collins, 2002], and max-margin markov networks [Taskaret *et al.*, 2003], and constrained conditional model. A number of machine learning problems involve learning from structured instances. One of the most important problem among them is sequence labeling. A lot of learning applications involve labeling and segmenting sequences. For example, if we have to do information extraction on some piece of text or identify genes in DNA. Figure 3.3(a) shows an example of information extraction problem as a sequence labeling task. Let $x = (x_1, \dots, x_T)$ represent the sequence on which information extraction is to be applied and $y = (y_1, \dots, y_T)$ be the sequence of labels that are given to each observation in the sequence. The labels specify whether a given word belongs to a particular entity class of interest (person,

organization and location) or not (null). For sequence-labeling problems like information extraction, labels are typically predicted by a sequence model based on a probabilistic finite state machine, such as the one shown in Figure 3.3(b)

x = Jake works in Calgary, Alberta with his brother Micheal.

y = person null null location location null null person person

(a)



(b)

Figure 3.3: (a) Information Extraction as Sequence Labeling (b) sequence model representing a finite state machine

The two important examples of structured output spaces classifiers are hidden markov models and structured support vector machines.

Hidden Markov Model (HMM)

The language models have been developed in the beginning of 20th Century when Andrei Markov used language models (Markov Models) to model letter sequences in works of Russian literature. Language models assign probabilities to strings of symbols. It assigns a probability to a piece of unseen text, based on some training data. These models are used for word prediction i.e. predicting the next word from the previous words by computing probability of the words. A language model assigns the probability to a sequence of m words $P(w_1, w_2, \dots, w_m)$ by means

of a probability distribution. It is used in many natural language processing applications such as speech recognition, machine translation, part-of-speech tagging, parsing and information retrieval, optical character recognition and data compression.

A Markov Model is a stochastic model that assumes the Markov Property. Markov Property refers to the memory less property of a stochastic/random process. A stochastic process has the Markov Property if the conditional probability distribution of future states of that process depends only upon the present state, not on the sequence of events that preceded it. Markov models are the class of probabilistic models that assume that we can predict the probability of some future unit without looking too far into the past i.e. the probability of the word depends only on the previous word [Jurafsky and Martin, 2008]. The simplest Markov model is the Markov Chain. It is a mathematical system that undergoes transitions from one state to another, between a finite or countable number of possible states. It is a random process characterized as memory less: the next state depends only on the current state and not on the sequence of events that preceded it.

Hidden Markov Model [Rabiner, 1989] is a Markov Chain for which the state is only partially observable. In other words, observations are related to the state of the system but they are typically insufficient to precisely determine the state. HMM is a statistical Markov Model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. An HMM can be considered as the simplest Bayesian network. In a regular Markov Model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In an HMM the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore, the sequence of tokens generated by an HMM gives some information about the sequence of states. In a Hidden Markov Model the word “hidden” refers to the state sequence through which the model passes, not to the parameters of the model. Even if the model parameters are known exactly the model is still hidden.

Structured Support Vector Machines (Structured SVM)

In machine learning, support vector machines are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis. SVM's are considered among the best supervised learning algorithms. In the

basic SVM the algorithm takes the inputs and makes the prediction about each input example and classifies it into one of the two possible classes. SVMs have been developed by Vapnik (1995) and are gaining popularity due to many attractive features, and promising empirical performance. Support Vector Machines for Classification and regression have been developed [Gunn, 1998]. SVM's have been shown as the maximum likelihood estimate of a class of probabilistic models [Franc *et al.*, 2011]. SVM's are intuitive, theoretically well- founded, and have shown to be practically successful. SVM's have also been extended to solve regression tasks (where the system is trained to output a numerical value, rather than yes/no classification) [Boswell, 2002].

The structured support vector machine [Nawozin and Lampert, 2011] is a machine learning algorithm that generalizes the SVM classifier. SVM classifier is used for binary classification, multiclass classification and regression, and the structured SVM is used for allowing training of a classifier for general structured output labels. Generalization of multiclass Support Vector Machine learning has been proposed that involves features extracted jointly from inputs and outputs. The resulting optimization problem has been solved efficiently by a cutting plane algorithm that exploits the sparseness and structural decomposition of the problem. The versatility and effectiveness of the method have been demonstrated on problems ranging from supervised grammar learning and named-entity recognition, to taxonomic text classification and sequence alignment [Tsochantaridis *et al.*, 2004]. Structured SVM's have also been used for other natural language processing applications like speech recognition [Zhang and Gales, 2011]. Structured support vector machines (SVMs) have been examined for noise robust speech recognition and the features based on generative models have been used, which allows model-based compensation schemes to be applied to yield robust joint features. The performance of the approach has been evaluated on a noise corrupted continuous digit task: AURORA 2.

3.2.2. Learning Pipeline Models

Another example of a complex model is a pipeline model. It has been applied to a number of applications successfully. In pipelining, the overall process is divided into a sequence of classifiers in such a way that each stage of the pipeline uses the output of the previous stage as its input and determines the prediction. Pipelining is a process in which a complex task is divided into many stages that are solved sequentially. A pipeline is composed of a number of elements

(processes, threads, co routines, etc.), arranged in such a way so that the output of each element is fed as input to the next in the sequence. Many machine learning problems are also solved using a pipeline model. Pipelining plays a very important role in applying the machine learning solutions efficiently to various natural language processing problems. The use of pipelining results in the better performance of these systems. A number of natural language processing applications have been carried out using pipeline models e.g. information extraction [Yu and Lam, 2010], dependency parsing and named entity recognition [Bunescu, 2008], and so on.

For explaining the process of pipelining we will again take an example of entity extraction as in Section 3.2. We will consider a sentence as shown in Figure 3.4. In this case, instead of making several local predictions regarding both segmentation and classification for each word and assembling them into a global prediction, a pipeline model would first learn an entity identification (segmentation) classifier and use this as input into an entity labeling classifier, which is then assembled into a two stage pipeline system.

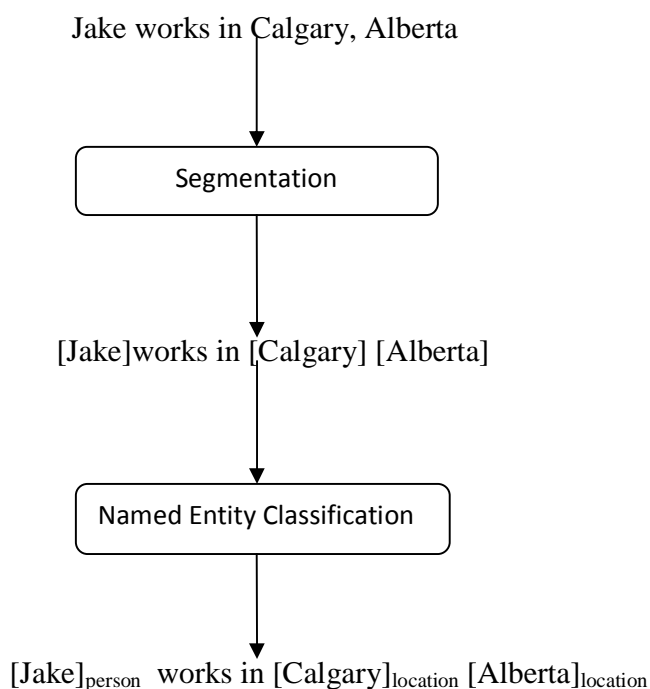


Figure 3.4: Pipelined Named Entity Recognition

The primary requirement of a pipeline model is that the feature vector generating procedure for each stage is able to use the output from previous stages of the pipeline, $\Phi^{(j)}(x, y^{(0)}, \dots, y^{(j-1)})$. To

train a pipeline model, each stage of a pipelined learning process takes m training instances $S^{(j)} = \{(\mathbf{x}_1^{(j)}, y_1^{(j)}), \dots, (\mathbf{x}_m^{(j)}, y_m^{(j)})\}$ as input to a learning algorithm $A^{(j)}$ and returns a classifier, $h^{(j)}$, which minimizes the respective loss function of the j th stage. Once each stage of the pipeline model classifier is learned, global predictions are made sequentially with the expressed goal of maximizing performance on the overall task, resulting in the prediction vector $\hat{y} = h(\mathbf{x}) = [\text{argmax } f_{y'}^{(j)}(\mathbf{x}^{(j)})]$ where $j=1$ to J and $y' \in Y^{(j)}$.

3.3. Pool-Based Active Learning

Until now we have been discussing supervised machine learning models. These models have been traditionally trained on whatever labeled data is made available to them. However, supervised methods have a number of disadvantages. One of the main disadvantages of using supervised methods is the high cost associated with them as they require large amounts of annotated data. Active learning [Settles, 2010] provides a way to reduce these labeled data requirements. These algorithms are capable of collecting new labeled examples for annotation by making queries to the expert. Active learning can reduce labeling effort required to train such models by allowing the learner to choose the instances from which it learns. There are different circumstances in which the learner may be able to ask queries. The learner may construct its own examples (membership query synthesis), request certain types of examples (pool-based sampling), or determine which of the unlabeled examples to query and which to discard (selective sampling). In active learning, the learner examines the unlabeled data and then queries only for the labels of instances which it considers to be informative. Therefore, an active learner learns only what it needs to in order to improve, thus reducing the overall cost of training an accurate system. Figure 3.6 [Settles, 2010] shows pool-based active learning.

In active learning the algorithm starts with a small number of labeled instances in the labeled training set L . It then requests the labels for a few carefully selected instances from the unlabeled pool U , learns from the query results, and then leverages its newly-found knowledge to choose which instances to query next. In this way, the active learner aims to achieve high accuracy using as few labeled instances as possible. There are many ways to select query instances, most of which stem from the uncertainty principle in experimental design and statistics [Federov, 1972]. One strategy for pool-based active learning is uncertainty sampling [Lewis and Gale, 1994]. It queries the instance that the model is least certain how to label. For probabilistic binary

classifiers, this means querying the instance $x \in U$ with the posterior probability $P(y = 1 \mid x; \theta)$ that is closest to 0.5 (i.e., the most ambiguous instance).

labeled training set

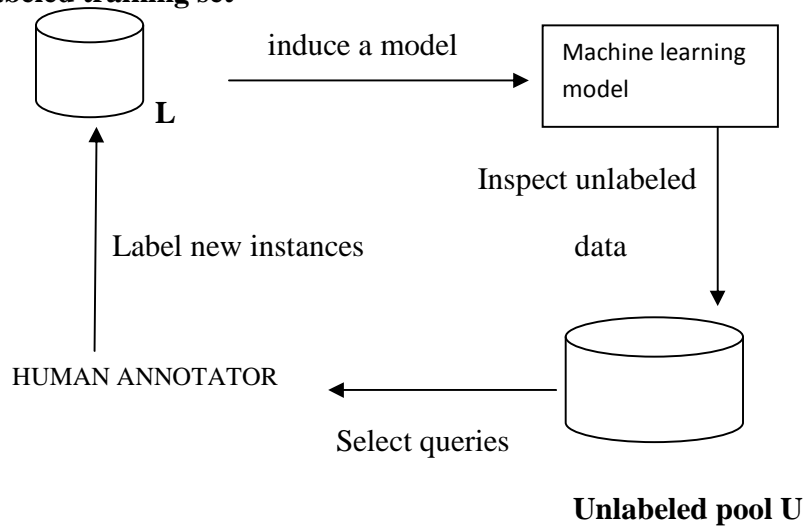


Figure 3.5: Pool-Based Active Learning

Chapter 4

Information Extraction and Machine Learning- A Pipelined Approach

In Section 3.2 of previous chapter we briefly discussed machine learning for complex models i.e. learning for structured instances and learning pipeline models. In this chapter, we discuss pipeline models in detail. As stated earlier, the main interest of this work is the use of machine learning techniques for natural language processing applications. Here we discuss the use of machine learning for an important natural language application i.e. information extraction. In Section 4.1 we provide an introduction about pipelining. In Section 4.2 we give a general overview of the information extraction process along with an example to show how the process will work. Section 4.3 discusses pipelining and machine learning and shows the steps of pipelining using active learning. In Section 4.4 we discuss stages of information extraction used in pipelining. In Section 4.5 we discuss various evaluation measures that are used to check the efficiency of machine learning models.

4.1. Introduction

Pipelining is a process in which a complex task is divided into many stages that are solved sequentially. A pipeline is composed of a number of elements (processes, threads, co routines, etc.), arranged in such a way so that the output of each element is fed as input to the next in the sequence. Many machine learning problems are also solved using a pipeline model. Pipelining plays a very important role in applying the machine learning solutions efficiently to various natural language processing problems. The use of pipelining results in better performance of these systems. However, these systems usually result in considerable computational complexity. A distinguishing feature of applications requiring pipeline models is that they often require significant quantities of labeled data to learn accurately, motivating the study of active learning in such scenarios. For this reason researchers were motivated for using active learning for these systems. Reason of using active learning is that these algorithms perform better than the traditional learning algorithms keeping the training data same. In this chapter we discuss an active learning strategy for pipelining of an important natural language processing task i.e. information extraction. The work described in this chapter has been previously published [Khan and Quadri, 2012a].

A number of natural language processing applications use machine learning algorithms. These applications include parsing, semantic role labeling, information extraction, etc. Using a machine learning algorithm for one natural language processing task often requires the output from

another task. Thus we can say these tasks are dependent on one another and therefore must be pipelined together. Therefore, a pipeline organization is used to model such situations. The benefit of using such an organization includes its ease of implementation and the main drawback is accumulation of errors between the stages of the pipeline that considerably affects the value of the results [Bunescu, 2008]. Pipelining has been used for a number of natural language applications e.g. bottom-up dependency parsing [Chang *et al.*, 2006], semantic role labeling [Finkel *et al.*, 2006]. A bidirectional integration of pipeline models has been developed as a solution to the problem of error accumulation in traditional pipelines [Yu and Lam, 2010]. In this chapter we show pipelining of information extraction. Although work has been done earlier in this regard which show pipelining of entity detection and relation extraction stages of information extraction. Here we theoretically discuss about including part-of-speech tagging stage of information extraction into the pipeline.

4.1.1. An Example of Pipelining

The primary motivation for modeling complex tasks as a pipelined process is the difficulty of solving such applications with a single classifier. For explaining the process of pipelining we will take an example of entity extraction as in Section 3.2. We will consider a sentence as shown in Figure 4.1. In this case, a pipeline model would first learn an entity identification (segmentation) classifier and use this as input into an entity labeling classifier, which is then assembled into a two stage pipeline system.

The primary requirement of a pipeline model is that the feature vector generating procedure for each stage is able to use the output from previous stages of the pipeline, $\Phi^{(j)}(x, y^{(0)}, \dots, y^{(j-1)})$. To train a pipeline model, each stage of a pipelined learning process takes m training instances $S^{(j)} = \{(x_1^{(j)}, y_1^{(j)}), \dots, (x_m^{(j)}, y_m^{(j)})\}$ as input to a learning algorithm $A^{(j)}$ and returns a classifier, $h^{(j)}$, which minimizes the respective loss function of the j th stage. Once each stage of the pipeline model classifier is learned, global predictions are made sequentially with the expressed goal of maximizing performance on the overall task, resulting in the prediction vector $\hat{y} = h(x) = [\text{argmax}_{y'} f^{(j)}(x^{(j)})]$ where $j=1$ to J and $y' \in Y^{(j)}$.

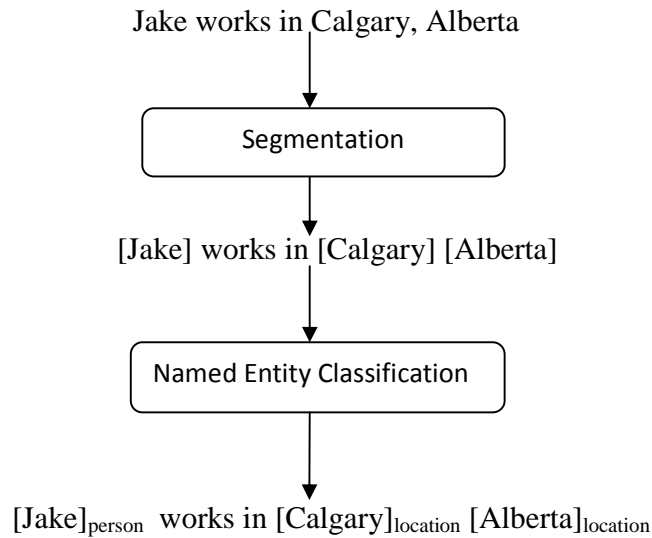


Figure 4.1: Pipelined Segmentation and Entity Detection

4.1.2. Why Active Learning

An important aspect of pipelined approaches is the corresponding high cost associated with obtaining sufficient labeled data for good learning performance. The active learning protocol minimizes this problem by allowing the learning algorithm to incrementally select unlabeled examples for labeling by the domain expert with the goal of maximizing performance while minimizing the labeling effort [Cohn *et al.*, 1996]. While receiving significant recent attention, most active learning research focuses on new algorithms as they relate to a single classification task. This work instead assumes that an active learning algorithm exists for each stage of a pipelined learning model and develops a strategy that jointly minimizes the annotation requirements for the pipelined process. In active learning the learning algorithm is capable of selecting additional instances to be labeled by maintaining access to the annotator. Thus active learning provides a way to reduce the labeling costs by labeling only the most useful instances for learning. Active learning reduces the amount of user effort required to learn a concept by reducing the number of labeled examples required [Arora and Agarwal, 2007]. In this learning technique, the learner is responsible for actively participating in the collection of the training examples i.e. obtaining the training set. The learner is capable of selecting a new input, observing the resulting output and including the new example based on the input and output into its training set. An important question that arises here is how to choose which input to try next

[Cohn *et al.*, 1996]. The learner uses some strategies for choosing the examples. The examples are chosen by making queries to the expert. The query strategy frameworks that have been used are uncertainty sampling [Lewis and Gale, 1994] and query-by-committee [Seung *et al.*, 1992].

4.2. Simple Architecture of Information Extraction

Information extraction (IE) can be defined as a process which involves automatic extraction of structured information such as entities, relationships between entities, and attributes describing entities from unstructured and/or semi-structured machine-readable documents [Sarawagi, 2008]. It can also be defined as a process of retrieving relevant information from documents. Applications of IE include news tracking [Turmo *et al.*, 2006], customer care [Bhide *et al.*, 2007], data cleaning [Sarawagi and Bhamidipaty, 2002], and classified ads [Michelson and Knoblock, 2005]. Figure 4.2 shows a simple architecture of information extraction system [Bird *et al.*, 2006]. The overall process of information extraction is composed of a number of subtasks such as segmentation, tokenization, part of speech tagging, named entity recognition, relation extraction, terminology extraction, opinion extraction, etc.

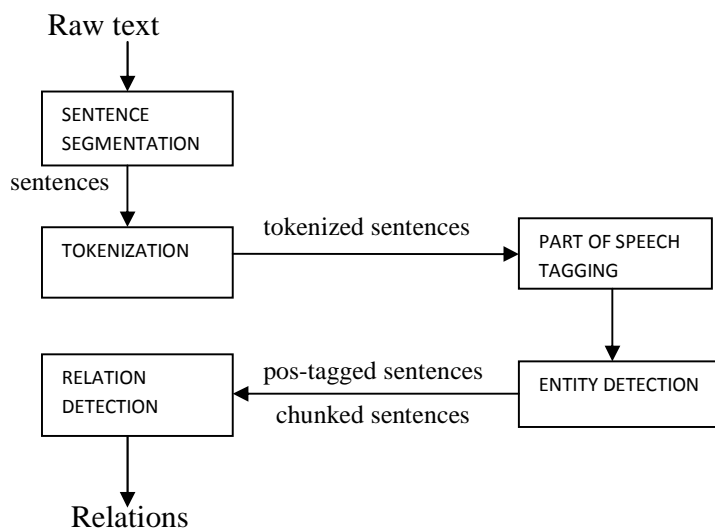


Figure 4.2: Simple Architecture of Information Extraction System

These subtasks of information extraction can be implemented using a number of different algorithms e.g. list-based algorithms for extracting person names or locations [Watanabe *et al.*,

4 INFORMATION EXTRACTION AND MACHINE LEARNING-A PIPELINED APPROACH

2009], rule-based algorithms for extracting phone numbers or mail addresses, and advanced machine learning and statistical approaches for extracting more complex concepts. Sentence segmentation is the process of breaking the text into component sentences. It is the process of determining the longer processing units consisting of one or more words. It consists of recognizing sentence boundaries between words in different sentences. Since most written languages have punctuation marks which occur at sentence boundaries, sentence segmentation is frequently referred to as sentence boundary detection, sentence boundary disambiguation, or sentence boundary recognition. All these terms refer to the same task: determining how a text should be divided into sentences for further processing. Tokenization breaks the text into meaningful elements such as words, symbols. It is the process of breaking up the sequence of characters in a text by locating the word boundaries, the points where one word ends and another begins. For computational linguistics purposes, the words thus identified are frequently referred to as tokens. In written languages where no word boundaries are explicitly marked in the writing system, tokenization is also known as word segmentation, and this term is frequently used synonymously with tokenization. This is followed by part-of-speech tagging which labels these tokens with their POS categories. An example of applying these steps to a piece of text is shown below in Figure 4.3

Jake works in Calgary, Alberta with his brother Micheal.

Jake	works	in	Calgary	Alberta	with	his	brother	Micheal
NP	VB	P	NP	NP	P	DET	NP	NP

Figure 4.3: Tokenization and Labeling

This is followed by entity detection. It is the process of identifying the entities having relations between one another, e.g. considering the above sentence, entities are detected as follows:

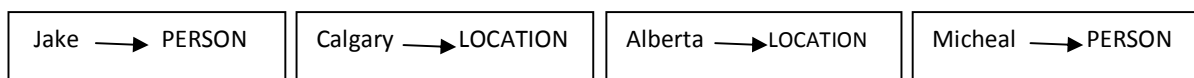


Figure 4.4: Entity Detection

Finally, after entities have been identified, the relations that exist between them are extracted in the relation detection step as follows:

{Jake, Calgary} → works_in

{Jake, Micheal} → brother_of

{Calgary, Alberta} → located_in

{Jake, Alberta} → works_in

Using pipelining in modeling the process of information extraction has resulted in an increase in efficiency. A lot of work has been done in this regard. Efficient information extraction pipelines have been developed that have resulted in the efficiency gains of up to one order of magnitude [Henning *et al.*, 2011]. A pipeline-based system has been developed for automated annotation of Surgical Pathology Reports [Kevin *et al.*, 2004]. There has been a lot of research in the field of information extraction using supervised machine learning. A number of supervised approaches have been proposed for the task of relation extraction which consists of some feature based methods [Kambhatla, 2004; Zhao and Grishman, 2005] and kernel methods [Lodhi *et al.*, 2002; Bunescu and Mooney, 2005]. However, supervised methods have a number of disadvantages. First of all, we cannot extend these methods to define new relations between the entities due to lack of new labeled data as supervised methods have a predefined set of labeled data. Same problem occurs if we wish to extend the entity relations to higher order. Also for large input data these methods are computationally infeasible [Bach and Badaskar, 2007]. One of the main disadvantages of using supervised methods is the high cost associated with them as they require large amounts of annotated data. Active learning [Settles, 2010] provides a way to reduce these labeled data requirements. These algorithms are capable of collecting new labeled examples for annotation by making queries to the expert. The main advantage of using pipelining is that when the pipelining process starts the examples that are selected first are those that are needed at the beginning phases of pipeline followed by those that are needed later.

4.3. Pipelining and Machine Learning

In the supervised machine learning problem a function maps the inputs to the desired outputs by determining which of a set of classes a new input belongs to. This is determined on the basis of the training data which contains the instances whose class is known e.g. classification problem. The mapping function can be represented by f , h denotes the hypothesis about the function to be learned. Inputs are represented as $X = (x_1, x_2, \dots, x_n)$ and outputs as $Y = (y_1, y_2, \dots, y_n)$ [Nilsson, 2005]. Therefore, hypothesis or the prediction function can be written as

$$h : X \rightarrow Y$$

h is the function of vector-valued input and is selected on the basis of training set of m input vector examples i.e.

$$\mathbf{X} = (x_1, x_2, \dots, x_n) \rightarrow \boxed{h} \rightarrow h(\mathbf{X})$$

$$\text{Training set} = \{ \mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_m \}$$

Therefore, the predicted value can be given as

$$y = h(x) = \operatorname{argmax}_{y' \in Y} f(x, y')$$

In case of pipelining, we have different stages. Let there be N stages. Therefore, each stage n depends on the previous $(n-1)$ stages i.e.

$$x, y^{(0)}, \dots, y^{(n-1)} \longrightarrow x^{(n)}$$

Therefore, in case of pipelining the predicted value can be written as

$$y = h(x) = [\operatorname{argmax} f^{(n)}(x^{(n)}, y')]$$

where $n = 1, \dots, N$.

As discussed earlier in this chapter, active learning algorithms reduce the number of labeled examples needed to learn any concept by collecting new unlabelled examples for annotation [Thompson *et al.*, 1999]. In active learning, the learner examines the unlabeled data and then queries only for the labels of instances which it considers to be informative. Therefore, an active learner learns only what it needs to in order to improve, thus reducing the overall cost of training

an accurate system. In active learning the algorithm starts with a small number of labeled instances in the labeled training set L . It then requests the labels for a few carefully selected instances from the unlabeled pool U , learns from the query results, and then leverages its newly-found knowledge to choose which instances to query next. In this way, the active learner aims to achieve high accuracy using as few labeled instances as possible. The examples are selected from the unlabelled data source U and are then labeled and added to the set of labeled data L [Settles, 2010]. Figure 4.5 shows the process of active learning [Settles, 2009]. The examples are selected by making queries to the expert. There are many ways to select query instances, most of which stem from the uncertainty principle in experimental design and statistics [Federov, 1972]. One strategy for pool-based active learning is uncertainty sampling [Lewis and Gale, 1994]. It queries the instance that the model is least certain how to label. For probabilistic binary classifiers, this means querying the instance $x \in U$ with the posterior probability $P(y = 1 | x; \theta)$ that is closest to 0.5 (i.e., the most ambiguous instance). Query strategies that have been used earlier are uncertainty sampling and query by committee [Seung *et al.*, 1992]. In both these strategies the point is to evaluate the informativeness of the unlabeled examples.

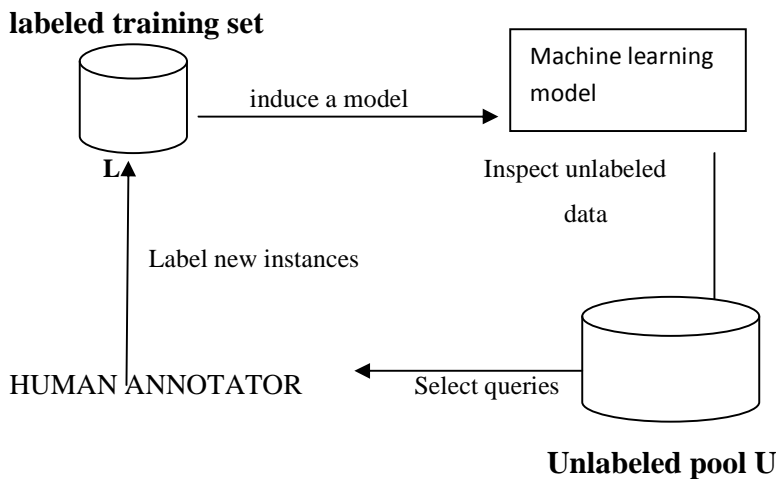


Figure 4.5: Process of Active Learning

The most informative instance or best query is represented as x_A^* , where A represents the query selection method used [Settles, 2010]. In uncertainty sampling, the algorithm selects that example about which it is least confident. In that case,

$$x_{LC}^* = \operatorname{argmax} 1 - P_{\theta}(y | x) \text{ [Culotta and McCallum, 2005]}$$

This approach is often straightforward for probabilistic learning models. For example, when using a probabilistic model for binary classification, an uncertainty sampling strategy simply queries the instance whose posterior probability of being positive is nearest 0.5 [Lewis and Gale, 1994; Lewis and Catlett, 1994]. For many learning algorithms, a widely used method of uncertainty sampling is to select instances for which their predicted label is least confident, either from a probabilistic viewpoint or through a margin-based analogue [Lewis and Gale, 1994; Tong and Koller, 2000; Schohn and Cohn, 2000; Culotta and McCallum, 2005; Roth and Small, 2006b; Settles and Craven, 2008].

In case of margin sampling,

$$x_{M}^* = \operatorname{argmin} P_{\theta}(y_1 | x) - P_{\theta}(y_2 | x) \quad (1)$$

where y_1 and y_2 are first and second most probable class labels [Scheffer *et al.*, 2001].

Another uncertainty sampling strategy that uses entropy as uncertainty measure,

$$x_{H}^* = \operatorname{argmax} - \sum_i P_{\theta}(y_i | x) \log P_{\theta}(y_i | x) \quad (2)$$

where y_i represents all the class labels [Settles, 2010]

The entropy-based approach can be generalized easily to probabilistic multi-label classifiers and probabilistic models for more complex structured instances, such as sequences [Settles and Craven, 2008] and trees [Hwa, 2004]. An alternative to entropy in these more complex settings involves querying the instance whose best labeling is the least confident:

$$\Phi_{LC}(x) = 1 - P(y^* | x),$$

where $y^* = \operatorname{argmax} P(y|x)$ is the most likely class labeling. This sort of strategy has been shown to work well, for example, with conditional random fields or CRFs [Lafferty *et al.*, 2001] for active learning in information extraction tasks [Culotta and McCallum, 2005; Settles and Craven, 2008].

Scoring functions are also used for selecting the examples to be labeled or annotated. Scoring functions are used for mapping an abstract concept to a numeric value. Here, the idea is to

calculate the score values for each instance to be labeled and the one with the minimum value is selected i.e.

$$x^* = \operatorname{argmin} q(x)$$

where x is selected from the unlabeled data U . The key difference between active learning and standard supervised learning is a querying function, which when provided with the data U and the learned classifier h returns a set of unlabeled instance from U . These selected instances are labeled and added to the supervised training set L used to update the learned hypothesis

Therefore, for each stage n of the pipeline, there is a separate querying function i.e. $q^{(n)}$, and after combining all these functions we get,

$$x^* = \operatorname{argmin} \Sigma q^{(n)}(x)$$

where $n = 1, \dots, N$ and x belongs to U and N is the total number of stages of a pipeline. The pipelining process using active learning consists of the following steps:

- a. As discussed earlier, each stage n of the pipeline has its own querying function $q^{(n)}$ and learner $l^{(n)}$. First of all, for each stage n , the hypothesis function as well as the querying function is estimated.
- b. The unlabelled examples or instances are then selected by the learner from unlabeled data U and after labeling are added to labeled data L for each stage n of the pipeline.
- c. As L changes after annotation of new instances, hypothesis is modified accordingly for each stage n .
- d. The process is repeated until the final hypothesis is obtained after all the N stages of pipeline have been completed.

4.4. Stages of Information Extraction used in Pipelining

Pipelining has been applied to information extraction earlier where the focus has been on entity detection and relation extraction. But as far as part-of-speech tagging is involved, not much has been done towards including it in the pipelining process of information extraction. Each stage of a pipeline is dependent on the earlier stages. In pipelining of information extraction, entity detection and relation detection highly depend on part-of-speech tagging. As discussed earlier,

part-of-speech tagging labels each word or phrase of a sentence with its POS category. It helps in recognizing different usages of the same word and assigns a proper tag e.g. in the sentences below the word “protest” has different usages:

The protest is going on. (Noun)

They protest against the innocent killings. (Verb)

Including part-of-speech tagging in the pipeline using active learning will result in the performance gain as the machine learning methods used for part-of-speech tagging have resulted in more than 95% accuracy. Moreover, in any natural language there are a number of words that are part-of-speech ambiguous (about more than 40%) and in such cases automatic POS tagging makes errors and hence require the use of machine learning techniques for tagging.

As discussed earlier, part-of-speech tagging labels each word or phrase of a sentence with its POS category, entity detection identifies the entities having relationships between one another in the sentence and relation detection extracts those relationships. Hence, in all these processes sentences are selected and annotated for all stages of the pipeline.

4.4.1. Including POS Tagging in Pipelining

Part-of-speech tagging (POS tagging), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic. E. Brill's tagger, one of the first and widely used English POS-taggers, employs rule-based algorithms. Different methods of POS tagging are Rule-Based POS tagging e.g., ENGTWOL [Voutilainen, 1995], transformation-based tagging e.g. Brill's tagger [Brill, 1995], and stochastic (probabilistic) tagging e.g. TNT [Brants, 2000]. POS tagging is used for a number of purposes e.g. it can help in determining authorship i.e. finding out are any two documents written by the same person (forensic linguistics) and it can help in speech synthesis and recognition. Labeling

natural language data with part-of-speech tags can be a complicated task, requiring much effort and expense, even for trained annotators. Several efforts, notably the Alembic workbench [Day *et al.*, 1997] and similar tools, have provided interfaces to aid annotators in the process. Automatic POS tagging of text using probabilistic models is mostly a solved problem but requires supervised learning from substantial amounts of training data. Previous work demonstrates the suitability of Hidden Markov Models for POS tagging [Kupiec, 1992; Brants, 2000]. More recent work has achieved state-of-the-art results with Maximum entropy conditional Markov models (MaxEnt CMMs, or MEMMs for short) [Ratnaparkhi, 1996; Toutanova& Manning, 2000; Toutanova *et al.*, 2003]. Part of the success of MEMMs can be attributed to the absence of independence assumptions among predictive features and the resulting ease of feature engineering.

In this section we theoretically show how active learning would be applied to POS tagging. As discussed earlier, first the informativeness of the unlabeled instances, sentences in our example, would be evaluated. Sentences would be selected from the unlabeled data and annotated/labeled by the annotator i.e. each word in the sentence would be tagged by its appropriate POS category. The annotated sentences will then be added to the labeled data. In Query By Uncertainty (QBU) approach, the informativeness of the unlabeled instances/examples is determined by evaluating the entropy- a measure of uncertainty associated with a random variable. In our example, these unlabeled instances are sentences. Therefore, we have to evaluate the entropy of sequence of words w_i in a sentence of length n , i.e.

$$H(w_1, w_2, \dots, w_n) = -\sum p(w_1, w_2, \dots, w_n) \log p(w_1, w_2, \dots, w_n)$$

From equation (2) we get,

$$x^*_H = -\sum p(y_i | x) \log p(y_i | x)$$

for each word w_i of the sentence, pos_i represents the part-of-speech tag for that word. Thus, the querying function for the part-of-speech tagging stage will be given as

$$q_{pos} = -\sum p(pos_i | w_i, y_i, pos_{i-1}, pos_{i-2}) \log p(pos_i | w_i, y_i, pos_{i-1}, pos_{i-2})$$

where $i = 1$ to n and pos_{i-1} and pos_{i-2} represent the tags of previous two words.

4.4.2. Active learning for Entity and Relation Detection

For this stage too QBU approach will be used which selects those unlabeled examples/instances about which the learner is least confident. According to equation (1), the best query in case of multi class uncertainty sampling is given by

$$x_M^* = \operatorname{argmin} P_\theta(y_1 | x) - P_\theta(y_2 | x)$$

where y_1 and y_2 are the first and second most probable class labels. Accordingly, the querying function for the entity and relation detection stage of information extraction can be given as

$$q_{\text{ERD}} = \operatorname{argmin} p(y | x_i) - p(y' | x_i)$$

or

$$q_{\text{ERD}} = \operatorname{argmin} [f(x_i, y) - f(x_i, y')]$$

$i = 1$ to n and y and y' are the first and second most probable class labels.

For all the stages, the performance would be calculated using three metrics i.e. precision, recall and F-measure. For POS tagging, precision would be calculated as number of correctly retrieved tags divided by the total number of retrieved tags. Recall would be calculated as number of correctly retrieved tags divided by the actual number of tags. For entity detection, precision would be calculated as the number of correctly extracted entities divided by the total number of extracted entities and recall would be calculated as number of correctly extracted entities divided by the actual number of entities. For relation extraction, precision would be calculated as the number of correctly extracted relations divided by the total number of extracted relations and recall would be calculated as the number of the correctly extracted relations divided by the actual number of relations. F- Measure for all these stages is equal to $2 * \text{precision} * \text{recall} / (\text{precision} + \text{recall})$.

4.5. Evaluation Measures

This section outlines various evaluation measures that are used for checking how well a model performs. For a particular label of interest, we are provided with a set of actual positives (e.g., objects that belong to that label) contained within the data set. The model then makes a set of predicted positives (e.g., the objects it assigns to that label) for the same data set. The actual and predicted label groupings can be thought of as indicator variables, and their cross product results

in four important values: tp (the number of true positives), fp (false positives), tn (true negatives), and fn (false negatives). Figure 4.6 [Settles, 2008] illustrates the relationship between these numbers. A basic evaluation measure is accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$. Basically, this measure represents the fraction of objects that the model labels correctly. In some problems, however, the data may be highly skewed, e.g., there might be nine times as many negative objects as positives. In a case like this, accuracy is a poor evaluation measure because a model that labels everything negative will still have accuracy = 0.9. In these situations, it is common instead to use precision, $P = \frac{tp}{tp+fp}$, the fraction of predictions that are correct, and recall, $R = \frac{tp}{tp+fn}$, the fraction of actual positives that are correctly predicted. Because of the inherent trade-off between precision P and recall R, a summary statistic called the F-Measure = $\frac{2 * P * R}{P+R}$ is commonly used when both are considered equally important. A final evaluation measure is the area under the Receiver Operating Characteristic (ROC) curve. An ROC curve measures the rate of true positives vs. false positives as a threshold is varied across a measure of confidence in its predictions (e.g., the model's posterior probability of the target label). It is regarded as a more appropriate measure than accuracy for some machine learning applications [Provost *et al.*, 1998]. The area under the curve AUROC, also called the Wilcoxon signed-rank test, can be interpreted as the probability that the model will rank a randomly chosen positive object higher than a randomly chosen negative.

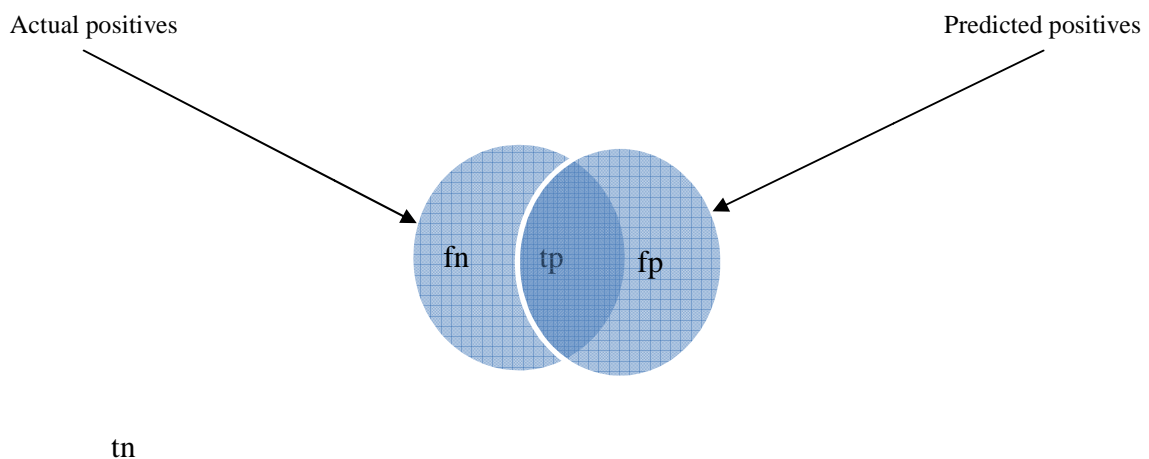


Figure 4.6: A Venn diagram illustrating the relationship between actual and predicted positives.

4 INFORMATION EXTRACTION AND MACHINE LEARNING-A PIPELINED APPROACH

The various overlaps define regions of tp (true positives), fp (false positives), tn (true negatives) and fn (false negatives).

Since it is trivial for a model to do well on the labeled data L that was used to train it, the practice of randomly partitioning data into a training set and an evaluation set is used, which do not overlap. In this way, the model is properly evaluated on new instances it has never seen before. To account for the effects of randomized partitioning, it is common to repeat an experiment for several runs and average the results. One particular way of doing this is cross-validation. In five-fold cross-validation, for example, the data is split into five partitions or folds. Then the five experiments are run for which each fold is held aside for evaluation, and the remaining four folds are used for training; then results are averaged across all folds.

Chapter 5

Evaluating Machine Learning Techniques for Efficiency

Machine learning is a vast field and has a broad range of applications including natural language processing, medical diagnosis, search engines, speech recognition, game playing and a lot more. A number of machine learning algorithms have been developed for different applications. However no single machine learning algorithm can be used appropriately for all learning problems. It is not possible to create a general learner for all problems because there are varied types of real world datasets that cannot be handled by a single learner. In this chapter we present an evaluation of various state-of-the-art machine learning algorithms using WEKA (Waikato Environment for Knowledge Analysis) for a real world learning problem- credit approval used in banks. Section 5.1 provides description about the components and working of WEKA. Section 5.2 describes the learning problem and the dataset that we have used in our experiments. In Section 5.3 we have explained the machine learning methods that we have evaluated. Section 5.4 provides description about our experimental setup and procedure and finally Section 5.5 shows the conclusion and the result. The work described in this chapter has been previously published [Khan and Quadri, 2012b].

5.1. Introduction

WEKA (<http://www.cs.waikato.ac.nz/ml/weka/>) is an open source software which consists of a collection of state-of-the-art machine learning algorithms and data preprocessing tools. It has been developed at the University of Waikato in New Zealand. It is designed in such a way that allows users to try all machine learning algorithms on new datasets easily. The WEKA system is written in Java. It can be used for a variety of tasks. It provides an implementation of state-of-the-art machine learning algorithms that we can apply to our datasets for extracting information about the data or we can apply several algorithms to our dataset for comparing their performance and choosing one for prediction. It also provides a number of tools for data preprocessing i.e. transforming datasets and analyzing the resulting classifier. Such tools are called filters. Thus the main focus of WEKA is on the learning methods and the filters. There are two ways in which we can invoke these methods: either by using command line options or by using the interactive graphical user interface. In our experiments we have used graphical user interface of WEKA because it is much more convenient. We have used WEKA 3.7.7.

5.1.1. WEKA- Interfaces

There are several ways by which we can access the functionality of WEKA. These are various interfaces and the simple CLI. Interfaces of WEKA include the Explorer, Experimenter and the Knowledge Flow.

5.1.1.1. Explorer

It is the most important graphical user interface in WEKA. Figure 5.1 shows the explorer interface. It consists of various tabs that are used for different tasks. First tab is the “Preprocess” tab. It is used for loading the datasets and transforming the datasets using filters. As shown in the figure datasets can be loaded as a file, from a URL or from databases using queries. WEKA allows files with specific formats e.g. ARFF, CSV, LibSVM’s format, and C4.5’s format.

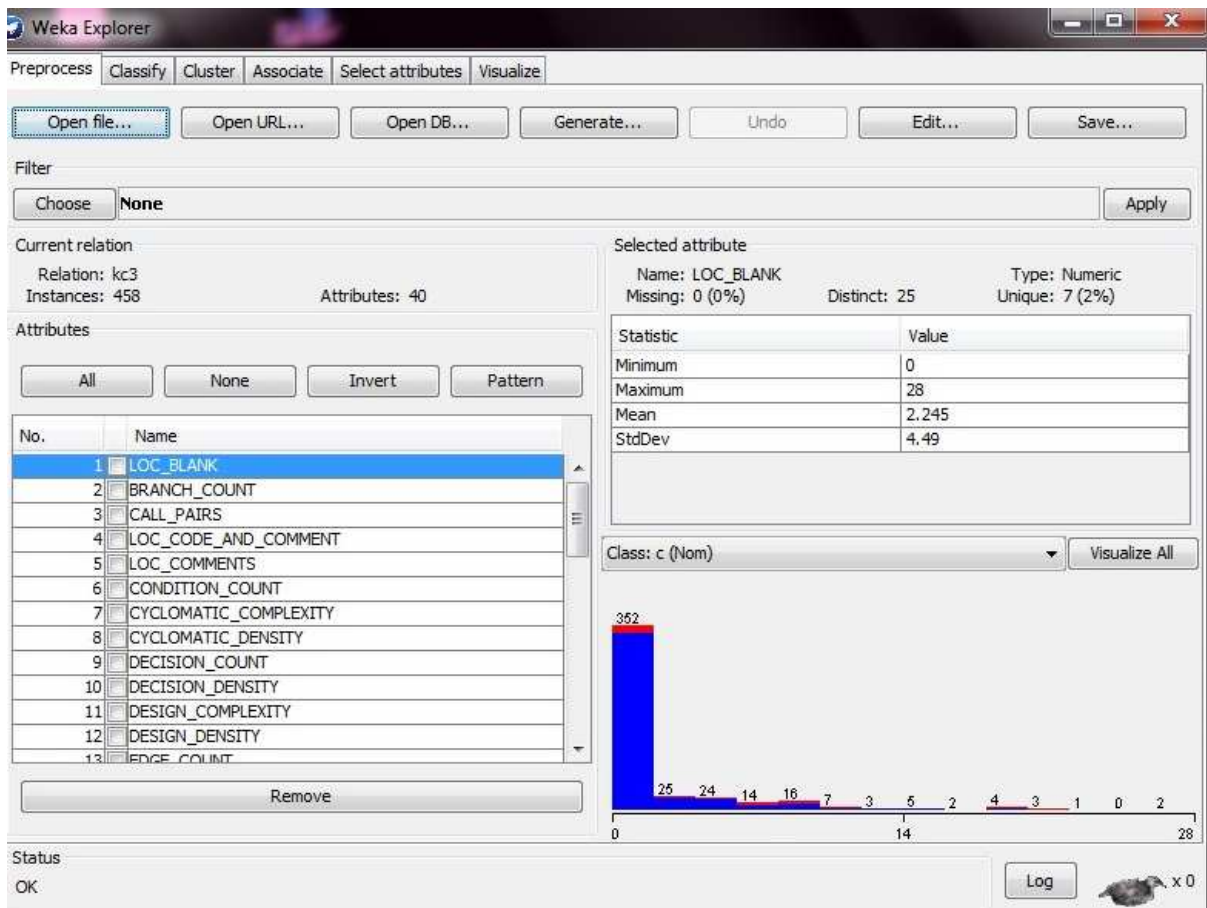


Figure 5.1: WEKA Explorer Interface showing Preprocess Tab

5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

After data is loaded it can be transformed by using various data preprocessing tools i.e. filters. Various discretization methods can be used for transforming these datasets or for dividing a dataset into training and testing sets using the appropriate filters.

Next is the “Classify” tab as shown in Figure 5.2. Through this tab we can use various classification and regression algorithms and applied to our preprocessed datasets. Classification algorithms typically produce decision trees or rules, while regression algorithms produce regression curves or regression trees. For a learning algorithm, the classify panel by default performs cross validation on the dataset that has been prepared in the Preprocess panel to estimate predictive performance. Other than cross-validation, test set can also be used. In that case we need to provide a test dataset separately. This panel also enables users to evaluate the resulting models, both numerically through statistical estimation and graphically through visualization of the data and examination of the model. This panel also allows us to visualize classifier errors, margin curve, threshold curve and so on. Moreover, it can visualize prediction errors in scatter plots, and also allows evaluation via ROC curves and other “threshold curves”. Models can also be saved and loaded in this panel.

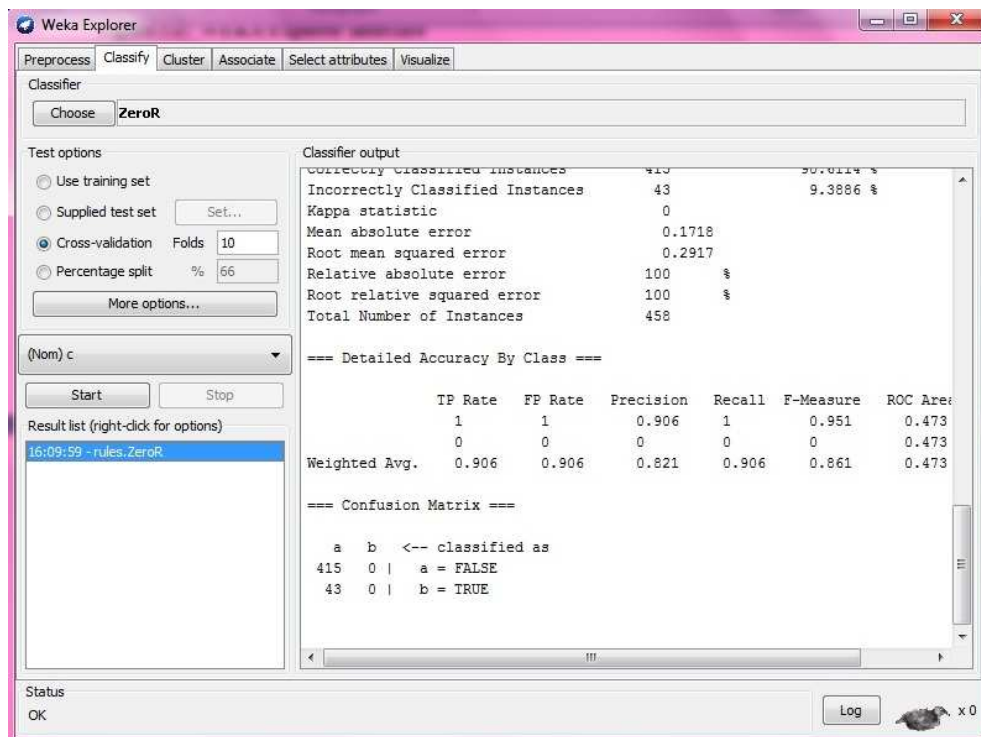


Figure 5.2: WEKA Explorer Interface showing Classify Tab

Apart from supervised classification algorithms, WEKA also provides unsupervised algorithms such as clustering and association algorithms. The third tab “Cluster” provides access to the clustering algorithms and the fourth tab “Associate” enables users to access algorithms for learning association rules. In the “Cluster” tab we can run a clustering algorithm on the data that has been loaded in the “Preprocess” panel.

The last two tabs are “Select attributes” and “Visualize”. “Select attributes” tab is used for identifying the most predictive attributes in the data. This tab has a lot of algorithms and evaluation criteria used for identifying the most important attributes in a dataset. It allows the users to access various methods for measuring the utility of attributes, and for finding attribute subsets that are predictive of the data. Robustness of the selected attribute set can be validated via a cross-validation-based approach.

Visualize tab is used for analyzing data visually. This presents a color-coded scatter plot matrix, and users can then select and enlarge individual plots. It is also possible to zoom in on portions of the data, to retrieve the exact record underlying a particular data point, and so on.

5.1.1.2. Experimenter

As shown in Figure 5.3, “Experimenter” is another interface of WEKA. As stated earlier, it is not possible to have a single machine learning method that works for all learning problems efficiently. Also there is no way to determine which learning method will work efficiently for a given problem at the beginning. For this purpose it is better to compare the performance of machine learning methods on various criteria. This interface is used for this purpose. Although it can also be done interactively in the “Explorer” interface, however “Experimenter” interface automates this process. This makes it easy to run the classification and regression algorithms with different parameter settings on a corpus of datasets, collect performance statistics, and perform significance tests on the results. Experiments can involve multiple algorithms that are run across multiple datasets; for example, using repeated cross-validation. Experiments can be saved in either XML or binary form. Saved experiments can also be run from the command-line. The Experimenter interface is not used much often by data mining practitioners as other WEKA’s interfaces. This interface makes identification of a suitable algorithm for a particular

dataset or collection of datasets easier once the initial experiments have been performed in the Explorer.

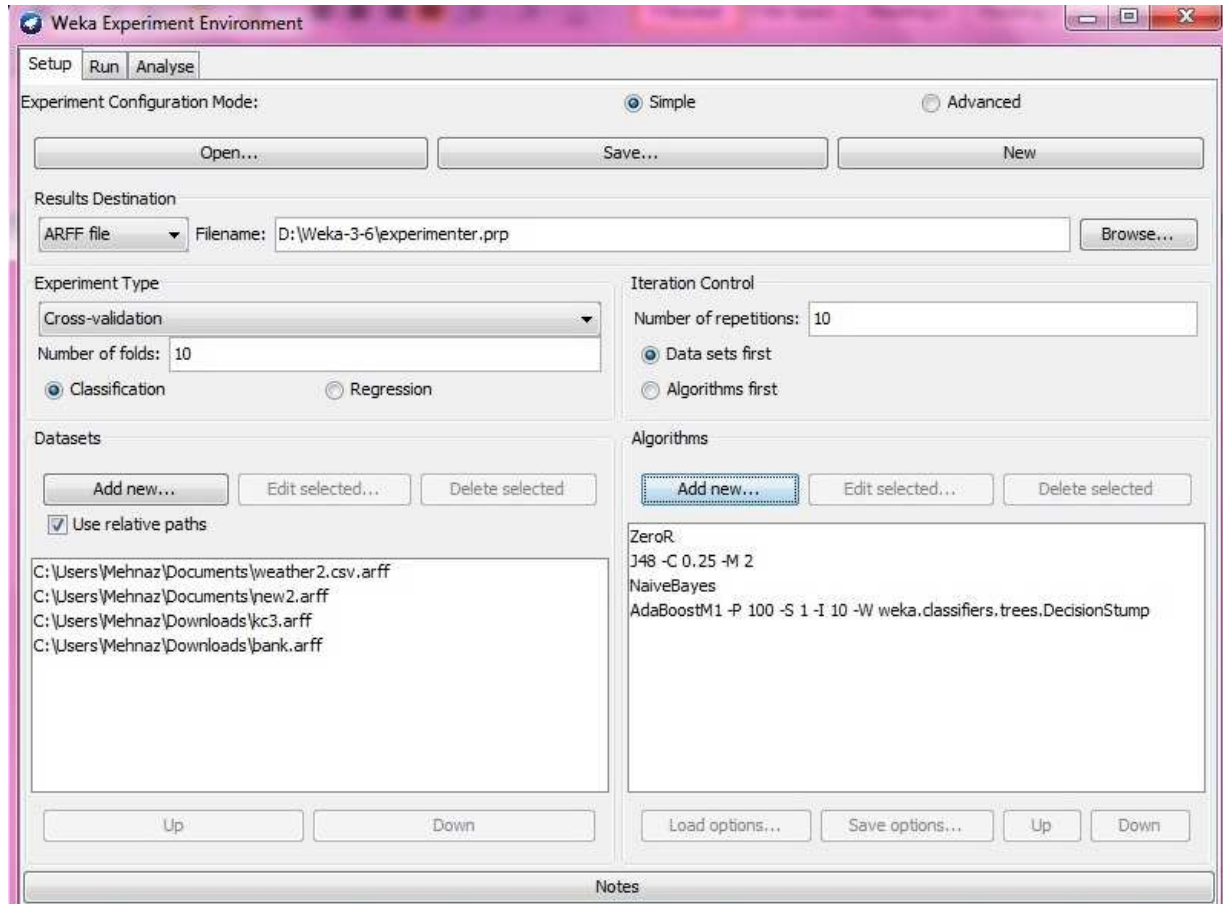


Figure 5.3: WEKA Experimenter Interface

5.1.1.3. Knowledge Flow

When we load a dataset in the “Explorer” interface, the entire dataset is loaded into the main memory for processing. It means that problems involving large datasets are not suitable for this method. In other words, “Explorer” interface does not allow for incremental learning and is only used for small to medium sized problems. However, some incremental algorithms are implemented that can be used to process very large datasets. One way to apply these is through the command-line interface, which gives access to all features of the system. An alternative, more convenient, approach is to use the second major graphical user interface, called “Knowledge Flow” which enables users to specify a data stream by graphically connecting components representing data sources, preprocessing tools, learning algorithms, evaluation

methods, and visualization tools. Its data flow model enables incremental updates with processing nodes that can load and preprocess individual instances before feeding them into appropriate incremental learning algorithms. It also provides nodes for visualization and evaluation.

5.1.2. Datasets

As stated in Section 1 of Chapter 1, the datasets used by machine learning algorithms consists of a number of instances that are represented using the same set of features. In supervised learning the instances are given with known labels (the corresponding correct outputs) in contrast to unsupervised learning, where instances are unlabeled. Table 5.1[Kotsiantis, 2007] shows instances with known labels.

Table 5.1: Example of a Dataset

Case	Feature 1	Feature 2	Feature n	Class
1	xxx	x		xx	Good
2	xxx	x		xx	Good
3	xxx	x		xx	Bad
...					...

WEKA applies its learning methods to a dataset and analyzes its output to extract information about the data. WEKA accepts the data in specific formats e.g. ARFF, CSV, LibSVM's format, and C4.5's format as stated earlier.

5.1.2.1. Preparing Datasets

The data that are has been collected for being used in the experiments can be stored anywhere e.g. in databases or spreadsheets. As we know WEKA supports some particular formats of data therefore we first need to convert the data into a suitable format before loading it in WEKA. The format we used for our experiments is ARFF format. The process of converting data into ARFF format is explained below.

Suppose we have our data in a spreadsheet program say MS Excel as shown in Figure 5.4. In order to convert it to ARFF format we first save it as a comma-separated file i.e. in CSV format. Then we load this CSV file in a text processor say MS Word as shown in Figure 5.5.

5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

id	age	sex	region	income	married	children	car	save_act	current_act	mortgage	pep
2	ID12101	48	FEMALE	INNER_CIT	17546	NO	1	NO	NO	NO	YES
3	ID12102	40	MALE	TOWN	30085.1	YES	3	YES	NO	YES	NO
4	ID12103	51	FEMALE	INNER_CIT	16575.4	YES	0	YES	YES	NO	NO
5	ID12104	23	FEMALE	TOWN	20375.4	YES	3	NO	NO	YES	NO
6	ID12105	57	FEMALE	RURAL	50576.3	YES	0	NO	YES	NO	NO
7	ID12106	57	FEMALE	TOWN	37869.6	YES	2	NO	YES	YES	NO
8	ID12107	22	MALE	RURAL	8877.07	NO	0	NO	NO	YES	NO
9	ID12108	58	MALE	TOWN	24946.6	YES	0	YES	YES	YES	NO
10	ID12109	37	FEMALE	SUBURBAN	25304.3	YES	2	YES	NO	NO	NO
11	ID12110	54	MALE	TOWN	24212.1	YES	2	YES	YES	YES	NO
12	ID12111	66	FEMALE	TOWN	59803.9	YES	0	NO	YES	YES	NO
13	ID12112	52	FEMALE	INNER_CIT	26658.8	NO	0	YES	YES	YES	YES
14	ID12113	44	FEMALE	TOWN	15735.8	YES	1	NO	YES	YES	YES
15	ID12114	66	FEMALE	TOWN	55204.7	YES	1	YES	YES	YES	YES
16	ID12115	36	MALE	RURAL	19474.6	YES	0	NO	YES	YES	YES
17	ID12116	38	FEMALE	INNER_CIT	22342.1	YES	0	YES	YES	YES	YES
18	ID12117	37	FEMALE	TOWN	17729.8	YES	2	NO	NO	NO	YES
19	ID12118	46	FEMALE	SUBURBAN	41016	YES	0	NO	YES	NO	YES
20	ID12119	62	FEMALE	INNER_CIT	26909.2	YES	0	NO	YES	NO	NO
21	ID12120	31	MALE	TOWN	22522.8	YES	0	YES	YES	YES	NO
22	ID12121	61	MALE	INNER_CIT	57880.7	YES	2	NO	YES	NO	NO
23	ID12122	50	MALE	TOWN	16497.3	YES	2	NO	YES	YES	NO
24	ID12123	54	MALE	INNER_CIT	38446.6	YES	0	NO	YES	YES	NO
25	ID12124	27	FEMALE	TOWN	15538.8	NO	0	YES	YES	YES	YES
26	ID12125	22	MALE	INNER_CIT	12640.3	NO	2	YES	YES	YES	NO
27	ID12126	56	MALE	INNER_CIT	41034	YES	0	YES	YES	YES	NO

Figure 5.4: Data in Excel spreadsheet

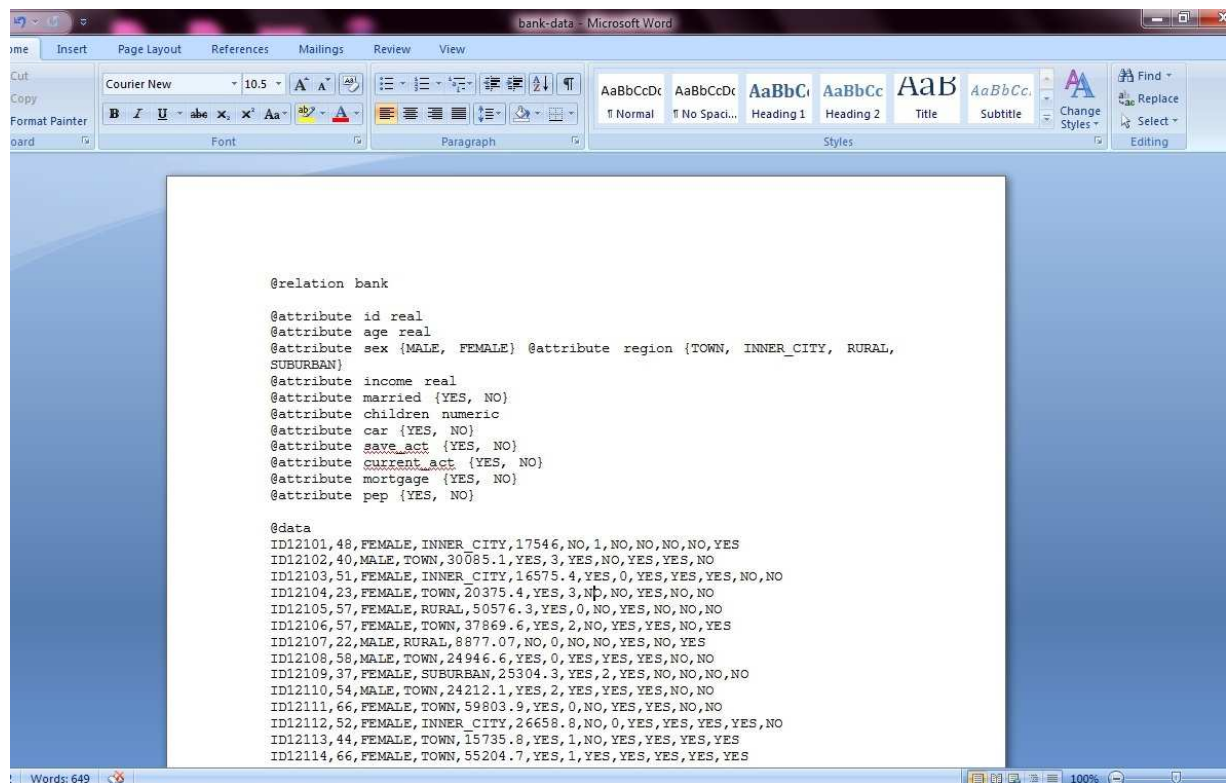
```

id, age, sex, region, income, married, children, car, save_act, current_act, mortgage, pep
ID12101, 48, FEMALE, INNER_CITY, 17546, NO, 1, NO, NO, NO, NO, YES
ID12102, 40, MALE, TOWN, 30085.1, YES, 3, YES, NO, YES, YES, NO
ID12103, 51, FEMALE, INNER_CITY, 16575.4, YES, 0, YES, YES, YES, NO, NO
ID12104, 23, FEMALE, TOWN, 20375.4, YES, 3, NO, NO, YES, NO, NO
ID12105, 57, FEMALE, RURAL, 50576.3, YES, 0, NO, YES, NO, NO, NO
ID12106, 57, FEMALE, TOWN, 37869.6, YES, 2, NO, YES, YES, NO, YES
ID12107, 22, MALE, RURAL, 8877.07, NO, 0, NO, NO, YES, NO, YES
ID12108, 58, MALE, TOWN, 24946.6, YES, 0, YES, YES, YES, NO, NO
ID12109, 37, FEMALE, SUBURBAN, 25304.3, YES, 2, YES, NO, NO, NO, NO
ID12110, 54, MALE, TOWN, 24212.1, YES, 2, YES, YES, YES, NO, NO
ID12111, 66, FEMALE, TOWN, 59803.9, YES, 0, NO, YES, YES, NO, NO
ID12112, 52, FEMALE, INNER_CITY, 26658.8, NO, 0, YES, YES, YES, YES, NO
ID12113, 44, FEMALE, TOWN, 15735.8, YES, 1, NO, YES, YES, YES, YES
ID12114, 66, FEMALE, TOWN, 55204.7, YES, 1, YES, YES, YES, YES, YES
ID12115, 36, MALE, RURAL, 19474.6, YES, 0, NO, YES, YES, YES, NO
ID12116, 38, FEMALE, INNER_CITY, 22342.1, YES, 0, YES, YES, YES, YES, NO
ID12117, 37, FEMALE, TOWN, 17729.8, YES, 2, NO, NO, NO, YES, NO
ID12118, 46, FEMALE, SUBURBAN, 41016, YES, 0, NO, YES, NO, YES, NO
ID12119, 62, FEMALE, INNER_CITY, 26909.2, YES, 0, NO, YES, NO, NO, YES
ID12120, 31, MALE, TOWN, 22522.8, YES, 0, YES, YES, YES, NO, NO
ID12121, 61, MALE, INNER_CITY, 57880.7, YES, 2, NO, YES, NO, NO, YES
ID12122, 50, MALE, TOWN, 16497.3, YES, 2, NO, YES, YES, NO, NO
ID12123, 54, MALE, INNER_CITY, 38446.6, YES, 0, NO, YES, YES, NO, NO
ID12124, 27, FEMALE, TOWN, 15538.8, NO, 0, YES, YES, YES, YES, NO
ID12125, 22, MALE, INNER_CITY, 12640.3, NO, 2, YES, YES, YES, NO, NO
ID12126, 56, MALE, INNER_CITY, 41034, YES, 0, YES, YES, YES, YES, NO
ID12127, 45, MALE, INNER_CITY, 20809.7, YES, 0, NO, YES, YES, YES, NO
ID12128, 39, FEMALE, TOWN, 20114, YES, 1, NO, NO, YES, NO, YES
    
```

Figure 5.5: Data after loading in MS Word

5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

In this file the rows of the original spreadsheet have been converted into lines of text, and the elements are separated from each other by commas. After that we have to convert the first line in which there are names of attributes into the header structure that makes up the beginning of an ARFF file. This is done by specifying the name of the dataset using @relation tag, the names, types, and values of each attribute are defined by @attribute tags, and @data tag is added before the data section of the file. This is shown in Figure 5.6.



```
@relation bank

@attribute id real
@attribute age real
@attribute sex {MALE, FEMALE} @attribute region {TOWN, INNER_CITY, RURAL,
SUBURBAN}
@attribute income real
@attribute married {YES, NO}
@attribute children numeric
@attribute car {YES, NO}
@attribute save_act {YES, NO}
@attribute current_act {YES, NO}
@attribute mortgage {YES, NO}
@attribute pep {YES, NO}

@data
ID12101,48,FEMALE,INNER_CITY,17546,NO,1,NO,NO,NO,NO,YES
ID12102,40,MALE,TOWN,30085.1,YES,3,YES,NO,YES,YES,NO
ID12103,51,FEMALE,INNER_CITY,16575.4,YES,0,YES,YES,YES,NO,NO
ID12104,23,FEMALE,TOWN,20375.4,YES,3,NO,NO,YES,NO,NO
ID12105,57,FEMALE,RURAL,50576.3,YES,0,NO,YES,NO,NO,NO
ID12106,57,FEMALE,TOWN,37869.6,YES,2,NO,YES,YES,NO,YES
ID12107,22,MALE,RURAL,8877.07,NO,0,NO,NO,YES,NO,YES
ID12108,58,MALE,TOWN,24946.6,YES,0,YES,YES,YES,NO,NO
ID12109,37,FEMALE,SUBURBAN,25304.3,YES,2,YES,NO,NO,NO,NO
ID12110,54,MALE,TOWN,24212.1,YES,2,YES,YES,YES,NO,NO
ID12111,66,FEMALE,TOWN,59803.9,YES,0,NO,YES,YES,NO,NO
ID12112,52,FEMALE,INNER_CITY,26658.8,NO,0,YES,YES,YES,YES,NO
ID12113,44,FEMALE,TOWN,15735.8,YES,1,NO,YES,YES,YES,YES
ID12114,66,FEMALE,TOWN,55204.7,YES,1,YES,YES,YES,YES,YES
```

Figure 5.6: Data after adding tags

After this we have to save this file with “Text Only with Line Breaks” as the file type. In this way, our data in spreadsheet gets converted into a format compatible with WEKA.

5.1.2.2. Training sets and Tests sets

In order to test the efficiency of our learning models we use training and test sets. We split our data into these two sets. The data used to construct or discover a predictive relationship are called the training data set. A test set is a set of data that is independent of the training data, but that follows the same probability distribution as the training data. The training set or the seen

data is used to build the model i.e. determine its parameters and the test set or the unseen data is used to measure its performance (holding the parameters constant). In supervised learning, the training set or the “gold standard” consists of both the input data as well as the correct/expected output i.e. the class values, and the test set is the data that we are going to apply to our method to test its efficiency. This set doesn’t have the output class values.

Sometimes another set called the validation set is also used in addition to training and test sets to tune the model. It is used to estimate how good your model has been trained. It cannot be used for testing.

5.1.2.3. Using the training and test sets in WEKA

WEKA allows us to use the dataset in a number of ways in our experiments. We can perform cross-validation, percentage split or we can use the supplied test set option. For using the “supplied test set” option we need to split our dataset into appropriate quantities of training and test sets. We first show how cross-validation works and then the process of splitting the dataset.

Cross-Validation

In cross-validation, mutually exclusive and same-sized subsets are created by dividing the training set. For each subset the classifier is trained on the union of all the other subsets. Using this technique the error rate of the classifier is calculated by the average of the error rate of each subset. WEKA allows us to specify how many folds we want to specify and usually we use 10 folds. In k-fold cross-validation, the data is randomly divided into k folds (subsets) of equal size. Then train the model on k-1 folds, use one fold for testing. This process is repeated k times so that all folds are used for testing. Finally, average performance is computed on the k test sets. This process helps in effectively using all the data for both training and testing [Keller, 2002].

Splitting the datasets

As stated earlier, for using supplied test set in WEKA we need to split our dataset into training and test sets. In the “Explorer” interface, we first load our dataset in the “Preprocess” panel. This is done either by loading an ARFF file or CSV file. We can also load our dataset directly from a URL or database. In our example, we have loaded the dataset using a URL as shown in Figure 5.7.

5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

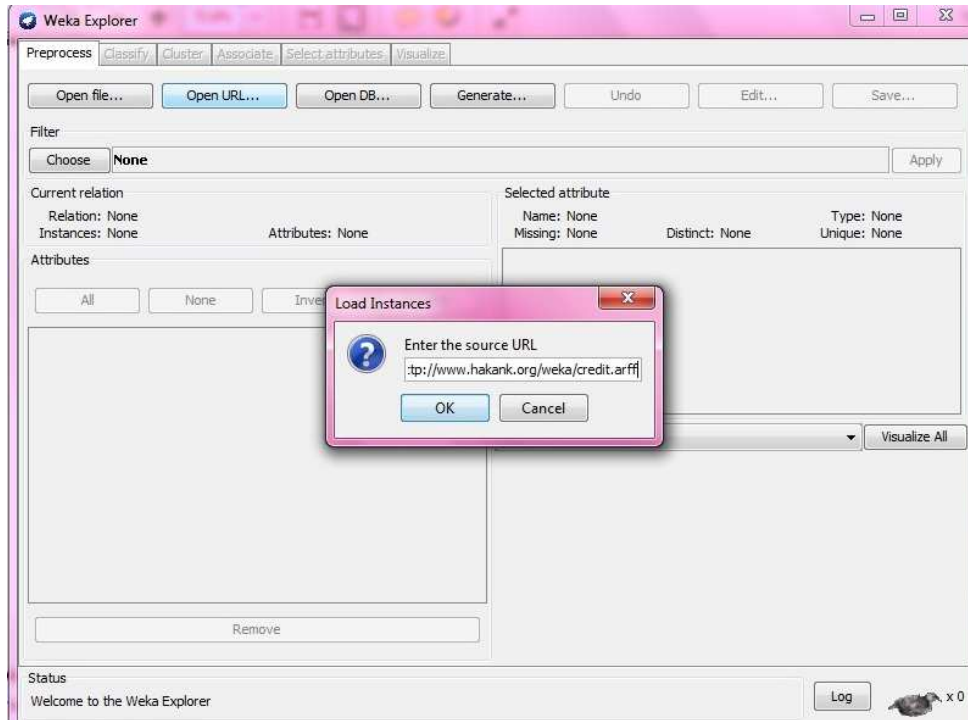


Figure 5.7: Loading Dataset from URL

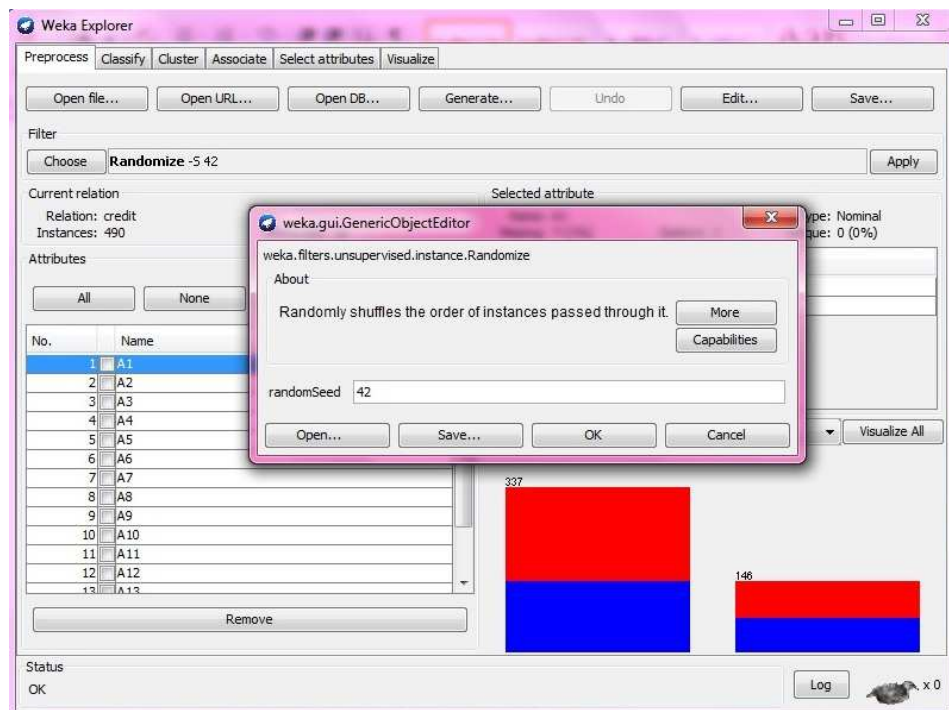


Figure 5.8: Using the Randomize filter

Next we have to split our dataset into two sets, 30% testing and 70% training. To do this we first randomize the dataset by choosing Randomize filter as shown in Figure 5.8. This creates a random permutation. Next we apply *RemovePercentage* filter on our dataset keeping percentage as 30 and we save the dataset as a training set. This is shown in Figure 5.9.

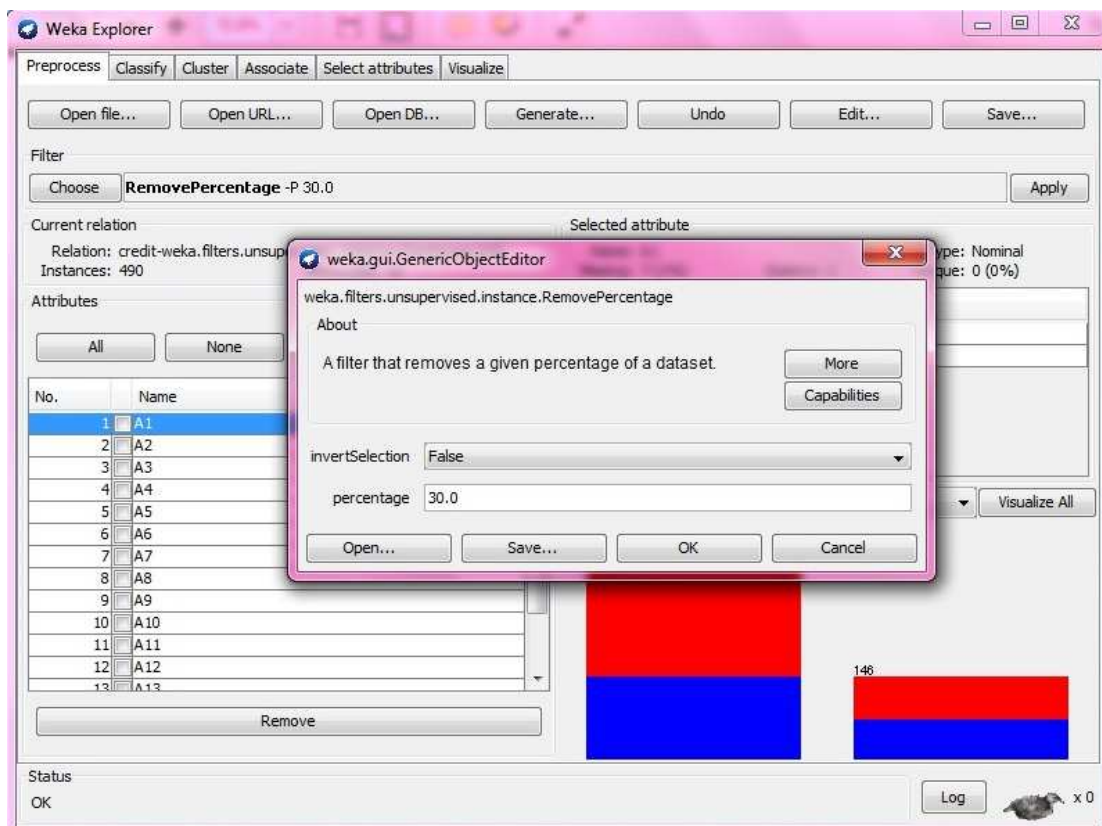


Figure 5.9: Using RemovePercentage filter.

Next we undo the change and again apply the same filter but changing the *invertSelection* option to “True” as shown in Figure 5.10. This picks the rest of the dataset i.e. 30% and is saved as a testing set.

This way our dataset gets divided into 30% testing and 70% training set. Next to use our sets in the experiments we choose the training set and move to the “Classify” panel and choose the procedure that we have to use and start the experiment. After that we apply the same procedure on our testing set to check what it predicts on the unseen data. For that, we select "supplied test set" and choose the testing dataset that we created. We run the algorithm again and we notice the differences in the confusion matrix and the accuracy.

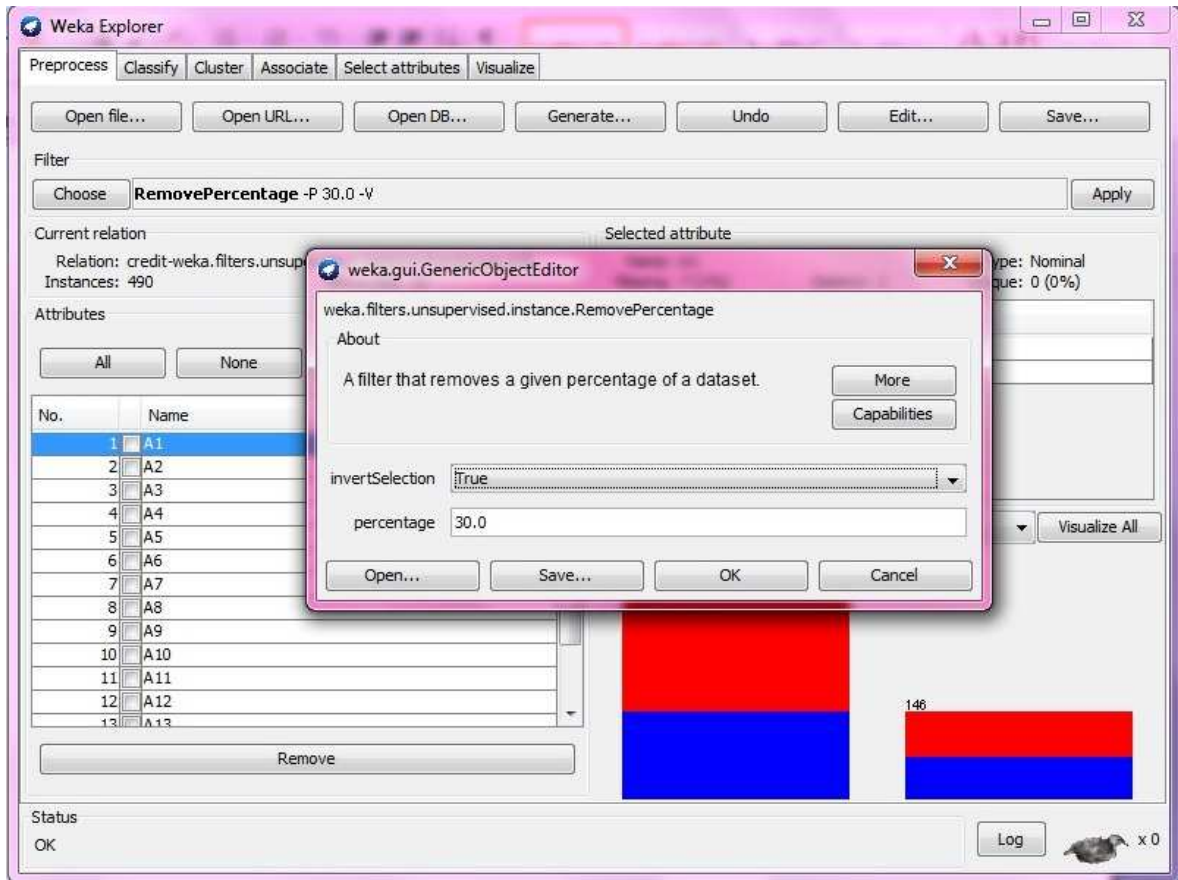


Figure 5.10: Using RemovePercentage filter with invertSelection

5.2. Learning problem and the Dataset used in our experiments

In our experiments we used credit approval problem used in banks for evaluating the efficiency of the state-of-the-art machine learning algorithms.

5.2.1. Understanding the problem

A financial institution, e.g. a bank, gives its customers an amount of money and expects them to pay it back in installments along with interest. This amount of money is called credit. However, before approving any credit application, it is necessary for the bank to make sure that the customer will pay the whole amount back. The bank should be able to predict in advance the risk associated with a loan. It is done for making sure that the bank will make a profit and that the customers get a loan within his or her financial capacity. This calls for a need to find out efficient

methods for automatic credit approval that can help the authorities in assessing credit applications effectively.

5.2.1.1. Risk involved in credit approval

Here the risk involved refers to the risk of loss to the financial institution if they lend the money to the customers who fail to pay the amount back [K.H. Ng, 1996]. The reason for this default can be anything like inability, unwillingness, etc. The bank should be able to predict in advance the risk associated with a loan. It is necessary for the lenders to calculate the probability of risk involved so that they can make correct decisions regarding the approval of the credit.

5.2.1.2. Credit evaluation method

Credit evaluation or credit scoring [Hand, 1998] is an evaluation system that is used for improving or increasing the abilities of the credit lenders in deciding the probability of the credit risk of a customer. In this method, risk is calculated by the bank on the basis of the amount of credit and the information about the customer. The information about the customer includes data that the bank has access to and is relevant in calculating financial capacity of the customer. This data consists of income, savings, collaterals, profession, age, past financial history, and so forth. The bank has a record of past loans containing such customer data and whether the loan was paid back or not. From this data of particular applications, the aim is to infer a general rule coding the association between a customer's attributes and his risk. That is, the machine learning system fits a model to the past data to be able to calculate the risk for a new application and then decides to accept or refuse it accordingly.

This process can be carried out in two ways. The first is called deductive credit scoring in which points are assigned to relevant customer attributes. These points are then used to form a credit score. The experience of the credit professionals is used to select the relevant attributes, determine the points and calculate the credit scores. Another type of credit scoring is empirical credit scoring in which the past data about the customers is analyzed and used to construct the scoring models. This is done by using appropriate algorithms for identifying characteristics relating to the credit risk of customers. These scoring models are then used to calculate the credit risk of new customers [Liu, 2001].

Bank professionals then use these credit scores to indicate the level of the credit risk and then decide accordingly whether to approve the credit to the customers or not and at what interest rate

the credit should be approved. For the low risk customers, the chances of getting the credit at lower interest rates and on longer repayment terms are higher. However, if the risk of the customers is high but lower than the cut-off credit risk, the customer is not disqualified from getting the credit but in this case the bank professionals review the customer application more carefully before deciding whether to approve or deny the credit request. If the credit is approved in case of such customers, it is given on higher interest rates and shorter repayment terms as compared to the low-risk customers.

5.2.1.3. Automating the process

The above stated processes i.e. credit scoring and approval can be carried out more efficiently if they are done automatically using computers. Automatic scoring and approval helps in gathering the necessary information quickly and speeds up the process of evaluation and determining whether to approve or deny credit applications. Automating this process does not mean that it can take place of the credit professional but it can help in making rapid decisions. The credit applications that are identified as good credit risk and those as bad credit risk may be automatically approved, or denied, while those of intermediate risk may still be passed to credit analysts for more detailed review before deciding whether to approve or deny credit. This can reduce the number of credit applications that need more detailed review and reduce the wastage of time, thus allowing credit analysts to concentrate only on those credit applications that are difficult or important.

5.2.2. Description of the Dataset used

The dataset (<http://www.hakank.org/weka/credit.arff>) that we used for our experiments for evaluating the learning algorithms was provided originally by Quinlan in his studies of ID3 and C4.5 system in 1987 and 1992, to induce decision trees for assessing credit card applications. It is the Australian Credit Approval dataset from UCI Repository of Machine Learning Databases and Domain theories (<http://archive.ics.uci.edu/ml/datasets.html>). The dataset consists of 15 attributes and a class label attribute. Before being made available to use, all the names and values of the attributes were changed to meaningless symbols to protect the confidentiality of the data. The values that the “class” attribute can take are + (positive) and – (negative). The attributes of the dataset are continuous, nominal with small numbers of values and nominal with larger numbers of values. The dataset consists of 490 instances with 44.5% being positive (credit approved), 55.5% being negative (credit denied) and 5% having missing values. Table 5.2 shows

the attribute names and attribute types of the dataset and Table 5.3 shows distributions of “+” and “-” values.

Table 5.2: Australian Credit Approval Dataset

Attribute	Type
A1	nominal
A2	continuous
A3	continuous
A4	nominal
A5	nominal
A6	nominal
A7	nominal
A8	continuous
A9	nominal
A10	nominal
A11	continuous
A12	nominal
A13	nominal
A14	continuous
A15	continuous
Class	nominal

Table 5.3: Class Distribution

Class	No. of instances
+	218
-	272

The “class” attribute can take two values i.e. “+” and “-” as stated earlier. The two values represent the low-risk and high-risk customers here. For low-risk customers, “class” attribute takes “+” value meaning credit can be approved for such customers and vice-versa for high-risk customers. This makes our learning problem a classification problem.

WEKA provides a number of classification algorithms that are accessible from the “Classify” tab as stated earlier. Hence our experiments use this dataset for evaluation of various classification learning algorithms. For our experiments we divided our dataset into training and test sets by the same procedure as described in Section 5.1.

5.3. Learning Methods Chosen For Evaluation

As discussed above, the learning problem that we have used in our experiments is a classification problem. Therefore, we have used WEKA’s classification algorithms for evaluation of the chosen dataset. In our experiments, we have chosen 10 learning algorithms from 6 different types. These are given below:

- Rule based
 - Zero R
 - One R
- Bayes Rule
 - NaiveBayes
 - NaiveBayesUpdateable
- Functions
 - Multilayer Perceptron
- Lazy Learners
 - KStar (K*)
- Tree Based
 - J48
 - RandomForest
- Meta-Algorithm
 - AdaBoostM1
 - Bagging

The sections that follow first explain each of these learners and then show their performance evaluation.

5.3.1. ZeroR and OneR

Both of these algorithms are rule-based algorithms. A rule-based algorithm uses rules to make deductions or choices. The classification method uses an algorithm to generate rules from the sample data. These rules are then applied to new data. OneR (One Rule) is a simple classification algorithm that generates a one-level decision tree. OneR classifier infers simple and accurate, classification rules from a set of instances. Performance studies of OneR classifier have shown that it produces rules that are only slightly less accurate than state-of-the-art learning schemes. It produces rules that are easy to interpret. OneR is also capable of handling missing values and numeric attributes showing adaptability despite simplicity. The OneR algorithm creates one rule for each attribute in the training data. It then selects the rule with the smallest error rate as its 'one rule'. It determines the most frequent class for each attribute value for creating a rule for an attribute. The most frequent class is simply the class that appears most often for that attribute value. A rule is simply a set of attribute values bound to their majority class; one such binding for each attribute value of the attribute the rule is based on. The error rate of a rule is the number of training data instances in which the class of an attribute value does not agree with the binding for that attribute value in the rule. OneR selects the rule with the lowest error rate. In the event that two or more rules have the same error rate, the rule is chosen at random. In the implementation of WEKA, the OneR algorithm picks the rule with the highest number of correct instances, not lowest error rate, and does not randomly select a rule when error rates are identical. Zero Regression (ZeroR) is a pseudo-regression method that always builds models with cross-validation coefficient $Q_2=0$. In the framework of this method the value of a property/activity is always predicted to be equal to its average value on the training set. This method is usually used as a reference point for comparing with other regression methods. ZeroR is the simplest classification method which relies on the target and ignores all predictors. ZeroR classifier simply predicts the majority category (class). Although there is no predictability power in ZeroR, it is useful for determining a baseline performance as a benchmark for other classification methods. The idea behind the ZeroR classifier is to identify the most common class value in the training set. It always returns that value when evaluating an instance. It is frequently used as a baseline for evaluating other machine learning algorithms.

5.3.2. NaiveBayes and NaiveBayesUpdateable

The Naive Bayes [Murphy, 2006] algorithm is based on conditional probabilities. It uses Bayes' Theorem. It is a formula that calculates a probability by counting the frequency of values and combinations of values in the historical data. Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. If B represents the dependent event and A represents the prior event, Bayes' theorem can be stated as follows.

$$\text{Prob}(B \text{ given } A) = \text{Prob}(A \text{ and } B) / \text{Prob}(A)$$

To calculate the probability of B given A, the algorithm counts the number of cases where A and B occur together and divides it by the number of cases where A occurs alone. A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature, given the class variable. An advantage of the naive Bayes classifier is that it only requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification.

NaiveBayesUpdateable is a class for a Naive Bayes classifier using estimator classes. This is the updateable version of NaiveBayes. This classifier will use a default precision of 0.1 for numeric attributes when buildClassifier is called with zero training instances.

5.3.3. MultiLayer Perceptron

It is a classifier that uses back propagation to classify instances. This network can be built by hand, created by an algorithm or both. The network can also be monitored and modified during training time. The nodes in this network are all sigmoid (except for when the class is numeric in which case the output nodes become unthresholded linear units). A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate output. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training the network. MLP is a modification of the standard

linear perceptron and can distinguish data that is not linearly separable. It is an artificial neural network generally used for classification or approximation. The MLP consists of a feed-forward network of neurons which map input vectors to output vectors. Each artificial neuron consists of a linear combination of weighted inputs which is passed through a non-linear activation function to produce the neuron's output. It is an extension of the perceptron in that it has at least one hidden layer of neurons. Layers are updated by starting at the inputs and ending with the outputs. Each neuron computes a weighted sum of the incoming signals, to yield a net input, and passes this value through its sigmoidal activation function to yield the neuron's activation value. Unlike the perceptron, an MLP can solve linearly inseparable problems [Steinwender and Bitzer, 2003].

5.3.4. J48 and Random Forest

Both these algorithms are decision tree based algorithms. A decision tree is a predictive machine-learning model that decides the target value (dependent variable) of a new sample based on various attribute values of the available data. The internal nodes of a decision tree denote the different attributes, the branches between the nodes tell us the possible values that these attributes can have in the observed samples, while the terminal nodes tell us the final value (classification) of the dependent variable.

The attribute that is to be predicted is known as the dependent variable, since its value depends upon, or is decided by, the values of all the other attributes. The other attributes, which help in predicting the value of the dependent variable, are known as the independent variables in the dataset. J4.8 algorithm is WEKA's implementation of C4.5 decision tree learner. C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan [Quinlan, 1993]. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. The J48 Decision tree classifier follows the following simple algorithm. In order to classify a new item, it first needs to create a decision tree based on the attribute values of the available training data. So, whenever it encounters a set of items (training set) it identifies the attribute that discriminates the various instances most clearly. This feature that is able to tell us most about the data instances so that we can classify them the best is said to have the highest information gain. Now, among the possible values of this feature, if there is any value for which there is no ambiguity, that is, for which the data instances falling within its category have the same value for

the target variable, then we terminate that branch and assign to it the target value that we have obtained. For the other cases, we then look for another attribute that gives us the highest information gain. Hence we continue in this manner until we either get a clear decision of what combination of attributes gives us a particular target value, or we run out of attributes. In the event that we run out of attributes, or if we cannot get an unambiguous result from the available information, we assign this branch a target value that the majority of the items under this branch possess.

Random forest is a powerful new approach to data exploration, data analysis, and predictive modeling. RandomForest implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points. Random forest (or random forests) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the classes output by individual trees. The algorithm for inducing a random forest was developed by Leo Breiman [Breiman, 2001] and Adele Cutler, and "Random Forests" is their trademark. The term came "from random decision forests" that was first proposed by Tin Kam Ho of Bell Labs in 1995. A random forest is a collection of CART-like trees following specific rules for tree growing, tree combination, self-testing, and post-processing [Steinberg *et al.*, 2004]. It is unexcelled in accuracy among current algorithms. It runs efficiently on large data bases. It can handle thousands of input variables without variable deletion. It gives estimates of what variables are important in the classification. It generates an internal unbiased estimate of the generalization error as the forest building progresses. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing. It has methods for balancing error in class population unbalanced data sets. Generated forests can be saved for future use on other data. Prototypes are computed that give information about the relation between the variables and the classification. It computes proximities between pairs of cases that can be used in clustering, locating outliers or (by scaling) give interesting views of the data. The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection. It offers an experimental method for detecting variable interactions.

5.3.5. KStar (K*)

K* is one of the lazy learning methods. Lazy learning methods or memory-based methods learn the structure of a domain by storing learning examples with their classification [Van den Bosch *et al.* 1996]. The domain model that results from a lazy learning process is able to generalize by using a predefined distance function. When the domain model is required to give the classification for an unseen domain element then it will use the distance function for finding the stored example that is closest to this unseen example. K* is an instance-based learner. Instance-based learners classify an instance by comparing it to a database of pre-classified examples. The fundamental assumption is that similar instances will have similar classifications. The question lies in how to define “similar instance” and “similar classification”. The corresponding components of an instance-based learner are the distance function which determines how similar two instances are, and the classification function which specifies how instance similarities yield a final classification for the new instance. In addition to these two components, IBL algorithms have a concept description updater which determines whether new instances should be added to the instance database and which instances from the database should be used in classification. For simple IBL algorithms, after an instance has been classified it is always moved to the instance database along with the correct classification. More complex algorithms may filter which instances are added to the instance database to reduce storage requirements and improve tolerance to noisy data [Cleary and Trigg, 1995]. K* is an instance-based classifier, that is the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function. It differs from other instance-based learners in that it uses an entropy-based distance function. The use of entropy as a distance measure has several benefits. Amongst other things it provides a consistent approach to handling of symbolic attributes, real valued attributes and missing values.

5.3.6. AdaBoostM1 and Bagging

Bootstrap aggregating (bagging) and boosting are useful techniques to improve the predictive performance of tree models. Boosting may also be useful in connection with many other models, e.g. for additive models with high-dimensional predictors; whereas bagging is most prominent for improving tree algorithms. Boosting is a general method for improving the performance of any learning algorithm. In theory, boosting can be used to significantly reduce the error of any

“weak” learning algorithm that consistently generates classifiers which need only be a little bit better than random guessing. Despite the potential benefits of boosting promised by the theoretical results, the true practical value of boosting can only be assessed by testing the method on “real” learning problems. AdaBoost [Freund and Schapire, 1996] is a boosting algorithm developed by Freund and Schapire that can be used to significantly reduce the error of any learning algorithm that consistently generates classifiers whose performance is a little better than random guessing.. AdaBoostM1 is a version of AdaBoost algorithm. Bagging is based on an idea of making various samples of the training set. A classifier is generated for each of these training set samples by a selected machine learning algorithm. In this way, for k variations of the training set we get k particular classifiers. The result will be given as a combination of individual particular classifiers.

5.4. Experimental Setup

In this section we show the performance evaluation of all the learning algorithms discussed above. We show their evaluation on the dataset chosen i.e. Credit Dataset. As already stated, we carried our experiments using WEKA. It provides a number of measures of evaluation that can be used to check the performance of the algorithms. When an experiment is run, results are displayed on “Classifier Output” area. This area has several sections showing different results. First is run information. It is a list of information giving the learning scheme options, relation name, instances, attributes and test mode that were involved in the process. After that classifier model (full training set) is displayed. It is a textual representation of the classification model that was produced on the full training data. Then a summary is shown that shows a list of statistics summarizing how accurately the classifier was able to predict the true class of the instances under the chosen test mode. A detailed accuracy by class, which is a more detailed per-class break down of the classifier’s prediction accuracy, is shown. Lastly, confusion matrix shows how many instances have been assigned to each class. Elements show the number of test examples whose actual class is the row and whose predicted class is the column. The evaluation measures that we used to compare the learners are number of correctly classified instances, time taken to build the model, F-Measure. For a particular label of interest, we are provided with a set of actual positives (e.g., objects that belong to that label) contained within the data set. The model then makes a set of predicted positives (e.g., the objects it assigns to that label) for the same data set.

The actual and predicted label groupings can be thought of as indicator variables, and their cross product results in four important values: tp (the number of true positives), fp (false positives), tn (true negatives), and fn (false negatives). A basic evaluation measure is accuracy = $tp+tn / tp+fp+tn+fn$. Basically, this measure represents the fraction of objects that the model labels correctly. In some problems, however, the data may be highly skewed, e.g., there might be nine times as many negative objects as positives. In a case like this, accuracy is a poor evaluation measure because a model that labels everything negative will still have accuracy = 0.9. In these situations, it is common instead to use precision, $P = tp / tp+fp$, the fraction of predictions that are correct, and recall, $R = tp / tp+fn$, the fraction of actual positives that are correctly predicted. Because of the inherent trade-off between precision P and recall R, a summary statistic called the F-Measure = $2 * P * R / P+R$ is commonly used when both are considered equally important.

Before using our dataset in the experiments, we used the method discussed in Section 5.1 for splitting it into 70% training set and 30% test set. First we loaded the actual dataset into the WEKA from URL (<http://www.hakank.org/weka/credit.arff>). Then after applying the splitting procedure, we saved both these sets as separate files, *trainingcredit.arff* and *testingcredit.arff*. For all experiments we used these two files. Figure 5.11 shows the actual dataset, Figure 5.12 shows *trainingcredit.arff* file and Figure 5.13 shows *testingcredit.arff* file.

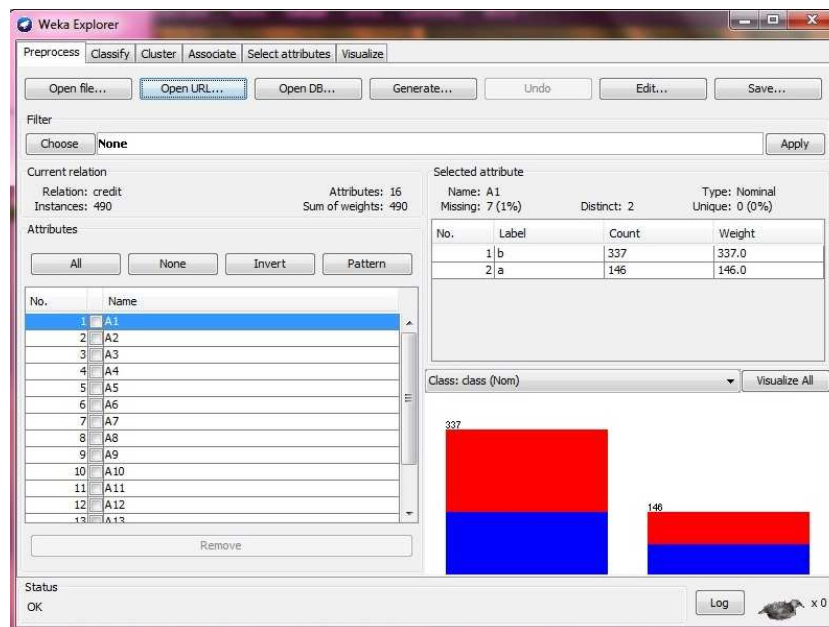


Figure 5.11: Credit Dataset

5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

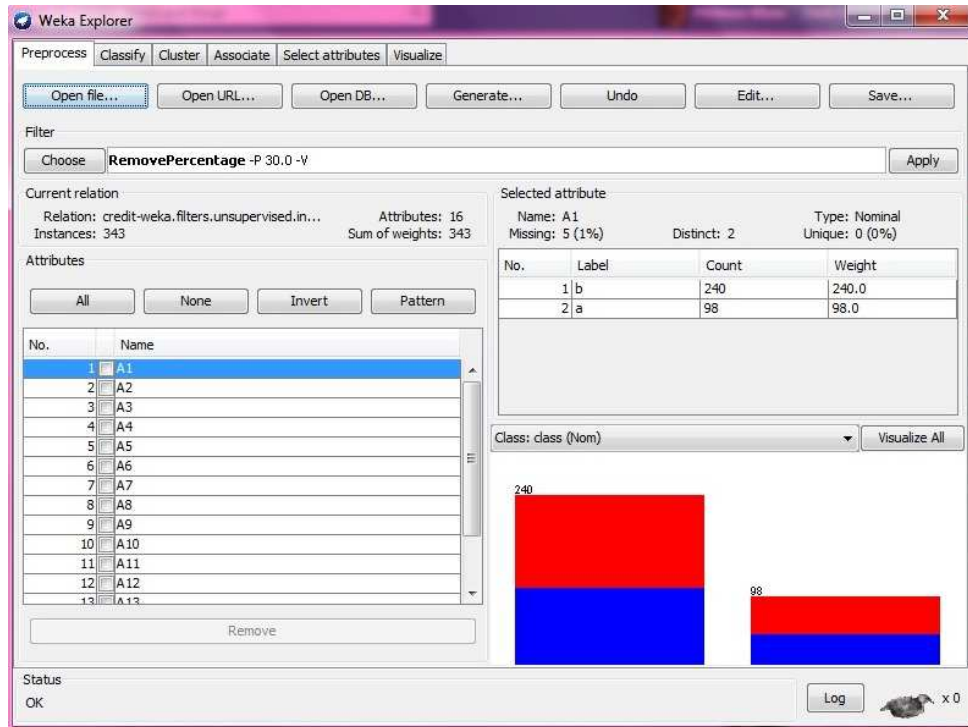


Figure 5.12: trainingcredit.arff file loaded in WEKA

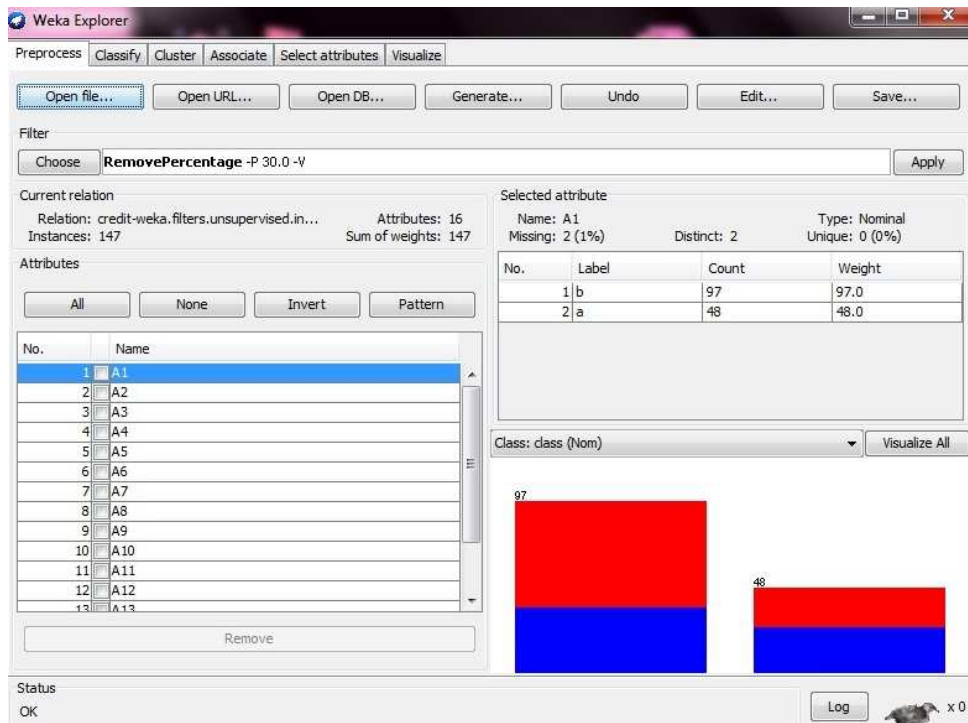


Figure 5.13: testingcredit.arff file loaded in WEKA

5.4.1. Experimental Procedure

- In our experiments, for each learner, we first load *trainingcredit.arff* file into WEKA through “Preprocess” panel.
- Then in the “Classify” panel we choose the classification algorithm to be implemented and start the analysis using 10-fold cross validation.
- After that we load the file *testingcredit.arff* using the “Supplied test set option” and then start the analysis again.
- Finally, we analyze the results on the basis of the evaluation measures discussed above.
- The same process is repeated for all the classification algorithms that are to be evaluated.

5.4.2. Experimental Results

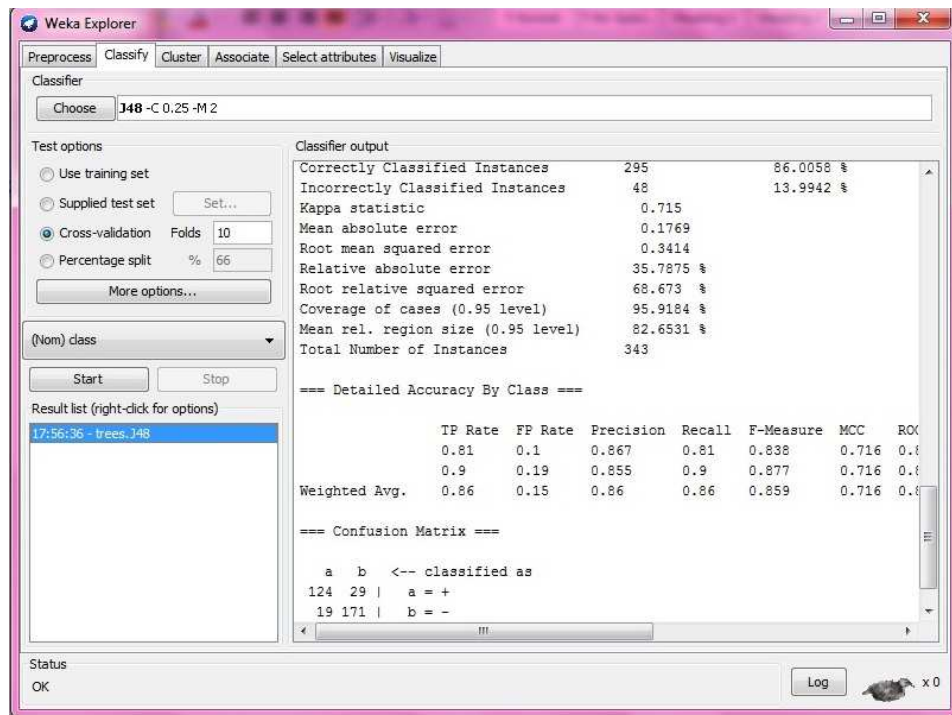


Figure 5.14: Results of J48 on trainingcredit.arff

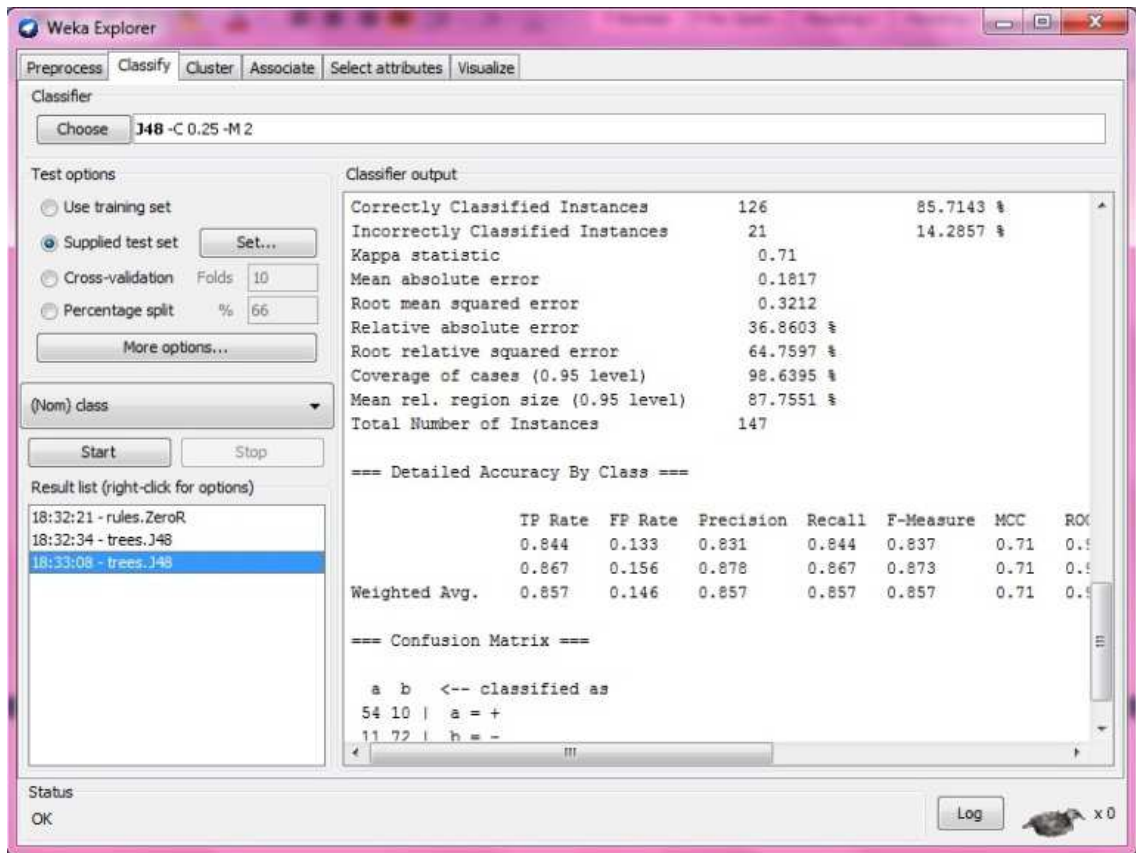


Figure 5.15: Results of J48 on testingcredit.arff

Results of J48:

Correctly Classified Instances (%) = 85.7143

Incorrectly Classified Instances (%) = 14.2857

Kappa Statistic = 0.71

Mean Absolute Error = 0.1817

F-Measure = 0.837

Time taken to build the Model = 0.01 seconds

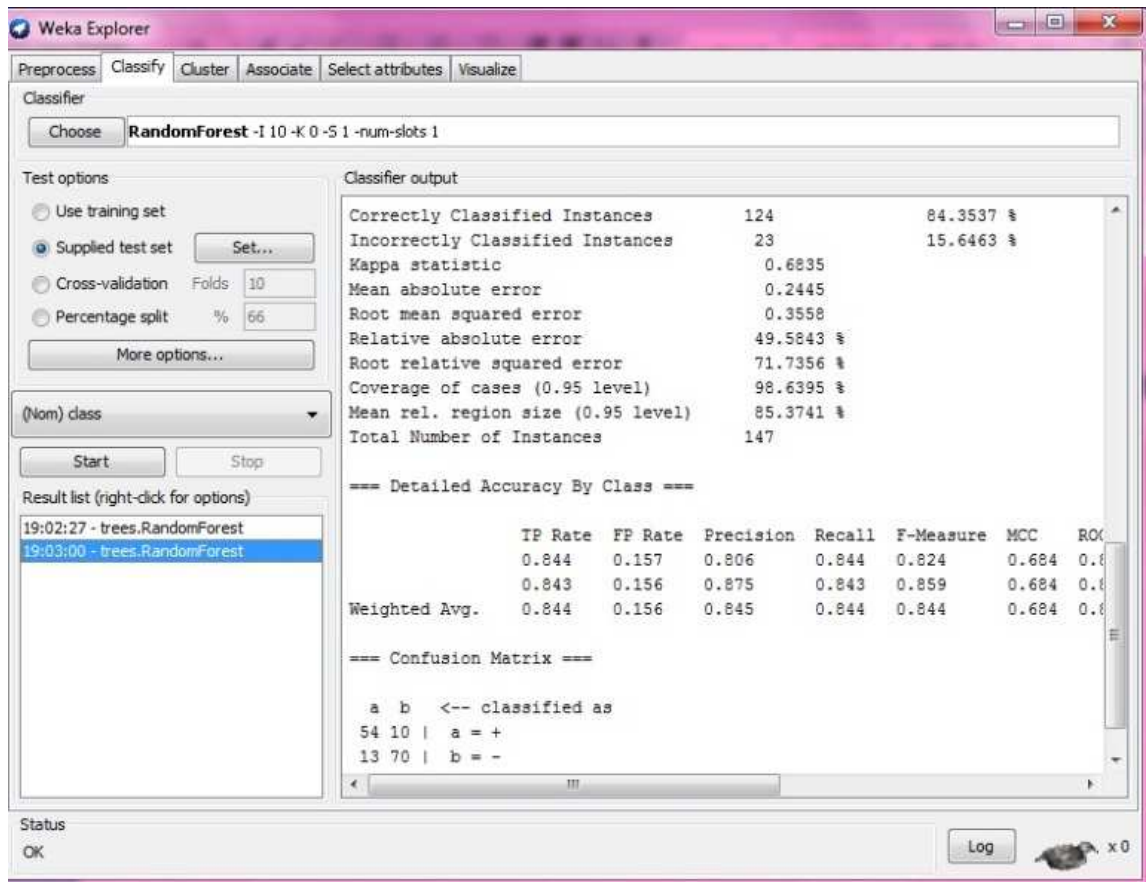


Figure 5.16: Results of RandomForest on testingcredit.arff

Results of RandomForest:

- Correctly Classified Instances (%) = 84.3537**
- Incorrectly Classified Instances (%) = 15.6463**
- Kappa Statistic = 0.6835**
- Mean Absolute Error = 0.2445**
- F-Measure = 0.824**
- Time Taken to build the Model = 0.05 seconds**

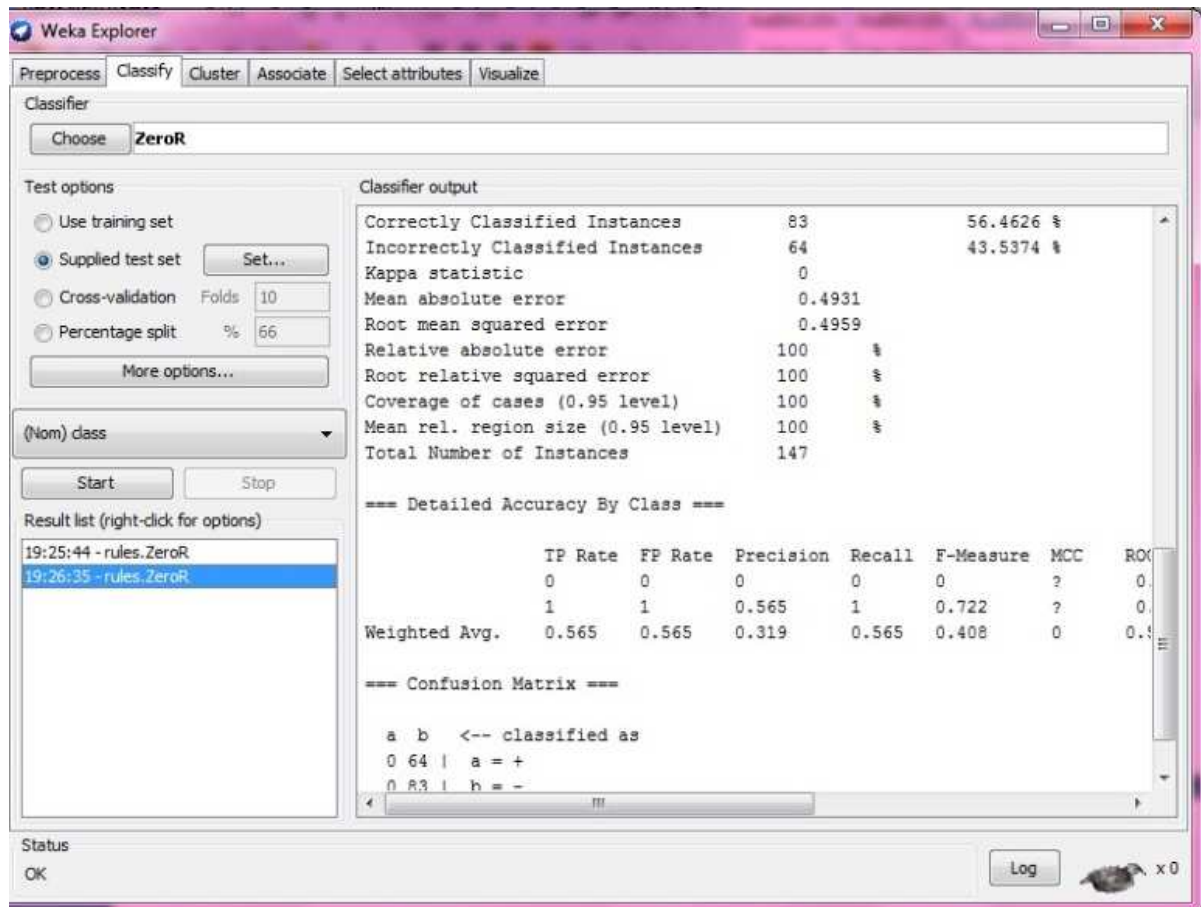


Figure 5.17: Results of ZeroR on testingcredit.arff

Results of ZeroR:

- Correctly Classified Instances (%) = 56.4626**
- Incorrectly Classified Instances (%) = 43.5374**
- Kappa Statistic = 0**
- Mean Absolute Error = 0.4931**
- F-Measure = 0**
- Time Taken to build the Model = 0 seconds**

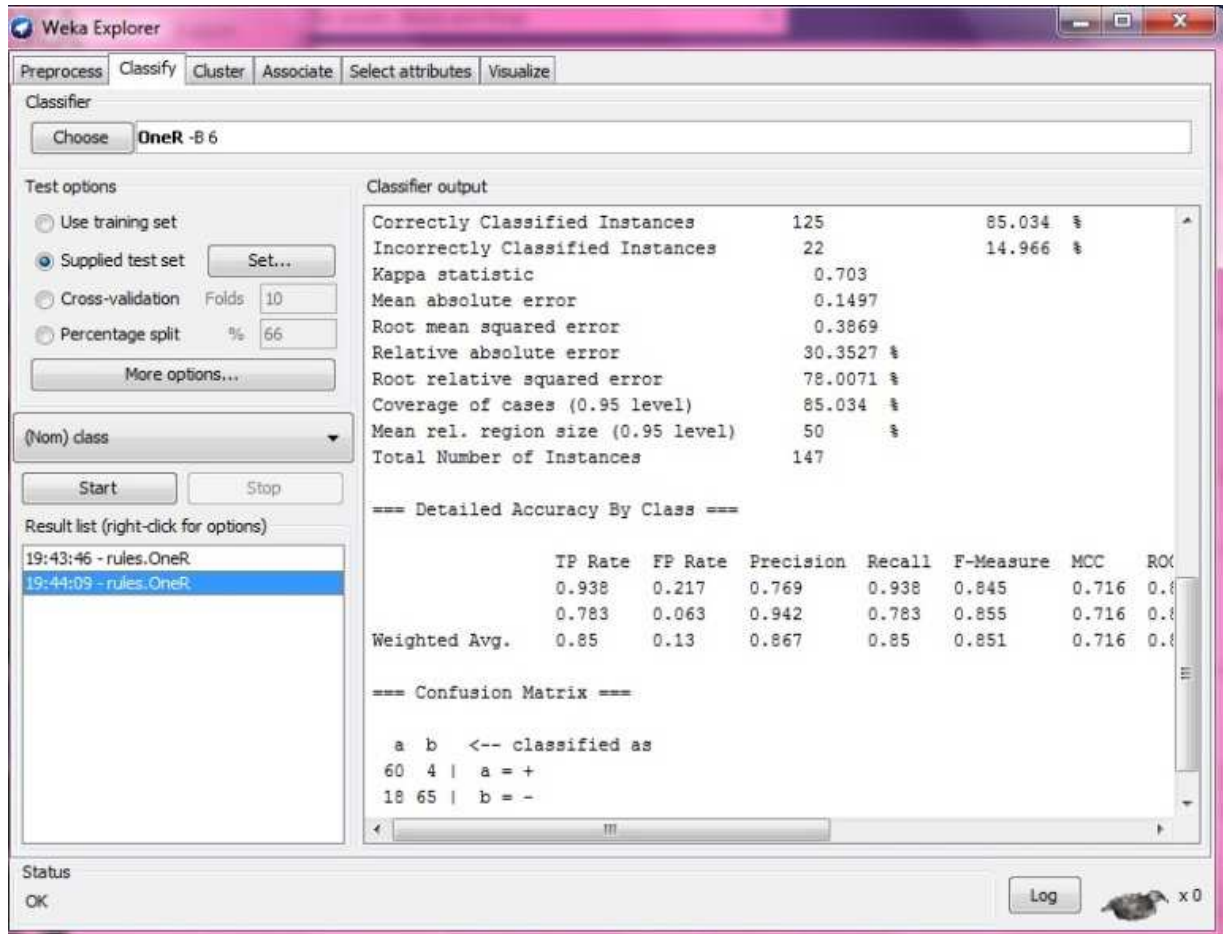


Figure 5.18: Results of OneR on testingcredit.arff

Results of OneR:

- Correctly Classified Instances (%) = 85.034**
- Incorrectly Classified Instances (%) = 14.966**
- Kappa Statistic = 0.703**
- Mean Absolute Error = 0.1497**
- F-Measure = 0.845**
- Time Taken to build the Model = 0 seconds**

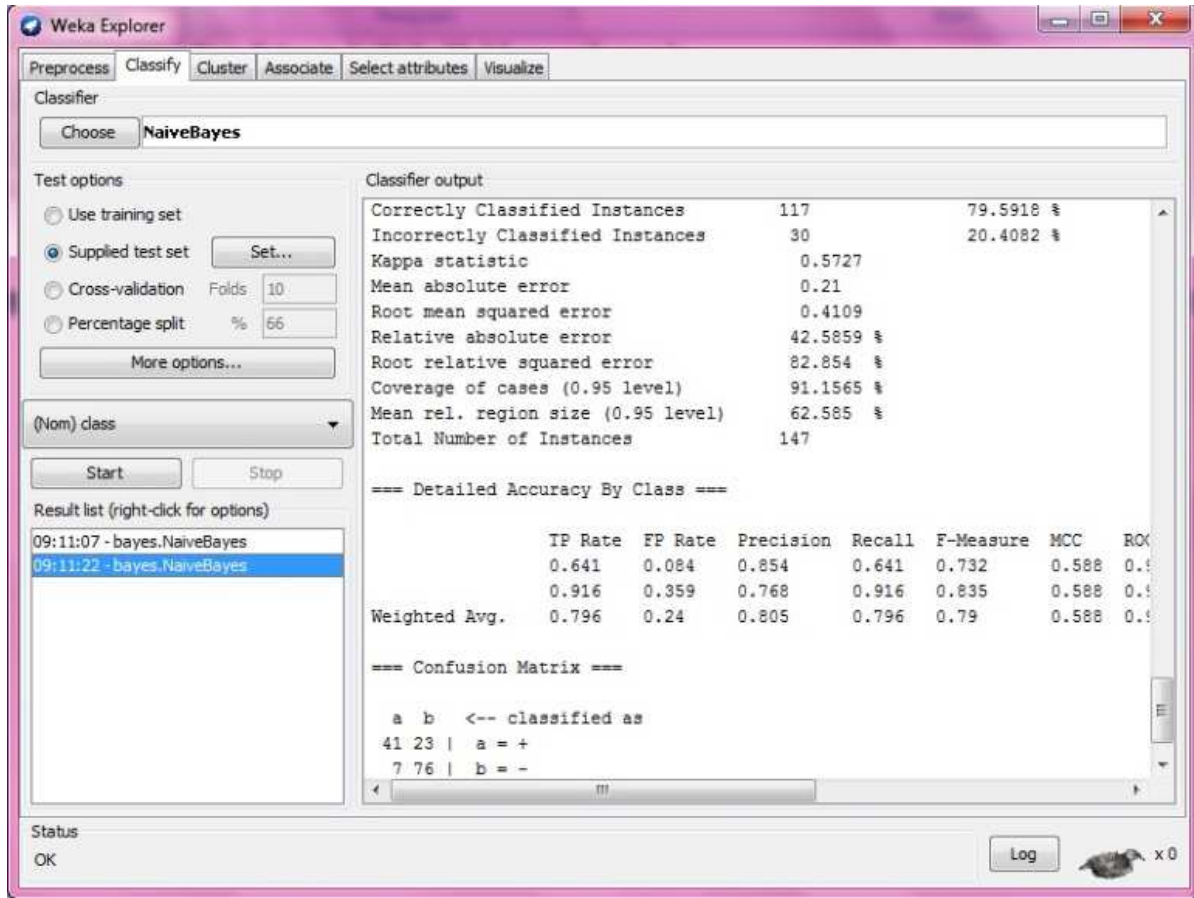


Figure 5.19: Results of NaiveBayes on testingcredit.arff

Results of NaiveBayes:

- Correctly Classified Instances (%) = 79.5918**
- Incorrectly Classified Instances (%) = 20.4082**
- Kappa Statistic = 0.5727**
- Mean Absolute Error = 0.21**
- F-Measure = 0.732**
- Time Taken to build the Model = 0.01 seconds**

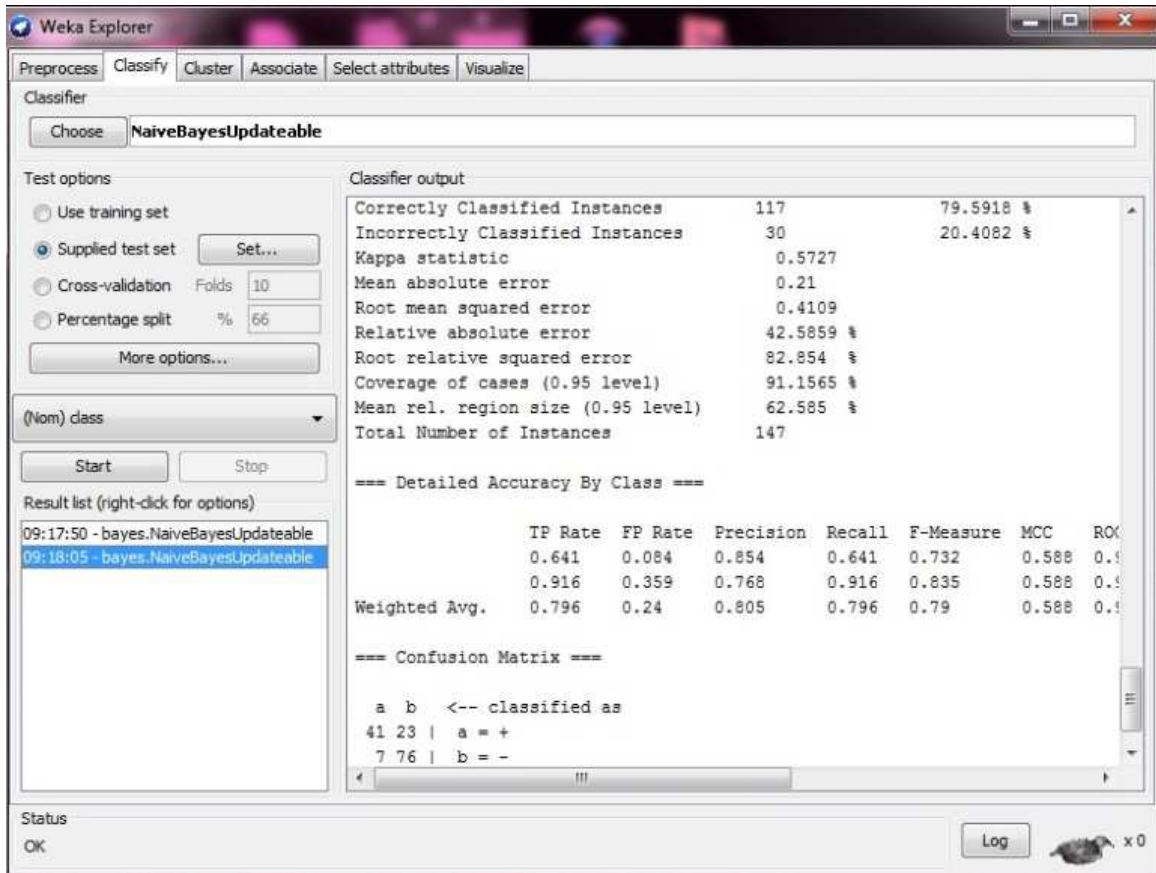


Figure 5.20: Results of NaiveBayesUpdateable on testingcredit.arff

Results of NaiveBayesUpdateable:

- Correctly Classified Instances (%) = 79.5918**
- Incorrectly Classified Instances (%) = 20.4082**
- Kappa Statistic = 0.5727**
- Mean Absolute Error = 0.21**
- F-Measure = 0.732**
- Time Taken to build the Model = 0.01 seconds**

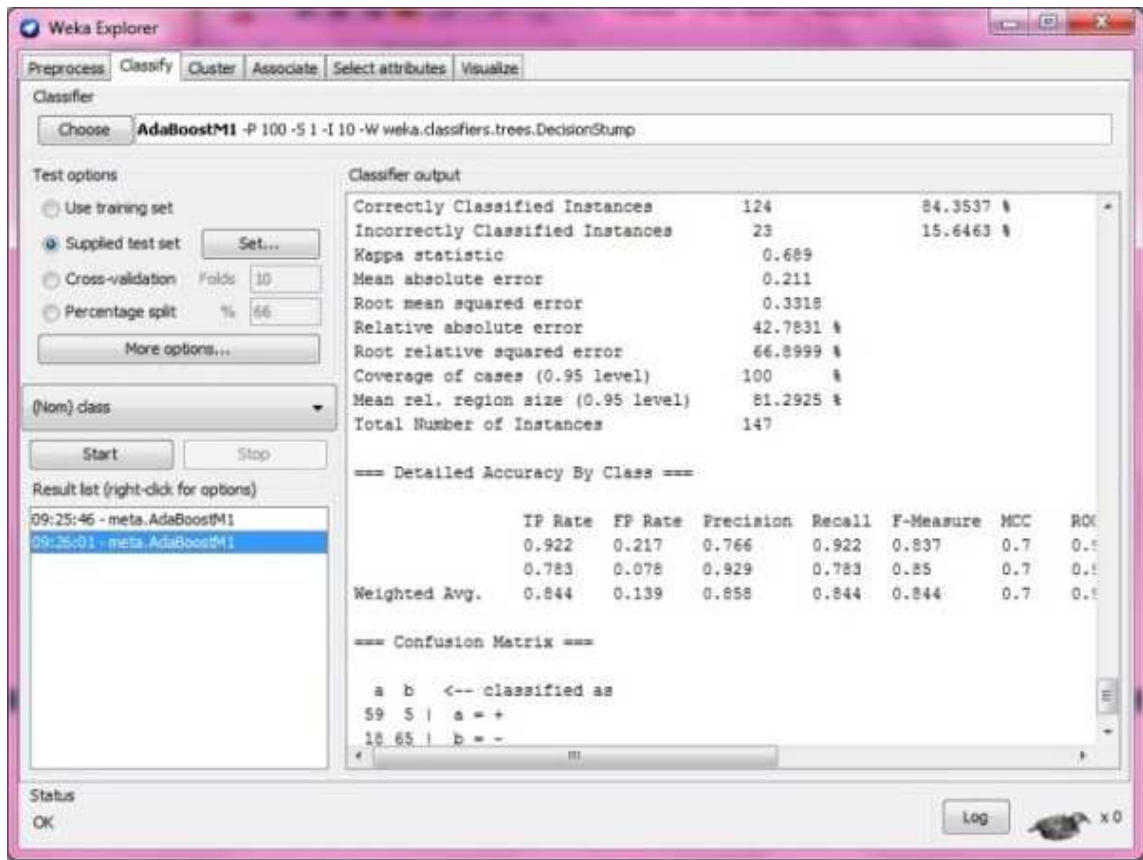


Figure 5.21: Results of AdaBoostM1 on testingcredit.arff

Results of AdaBoostM1:

- Correctly Classified Instances (%) = 84.3537**
- Incorrectly Classified Instances (%) = 15.6463**
- Kappa Statistic = 0.689**
- Mean Absolute Error = 0.211**
- F-Measure = 0.837**
- Time Taken to build the Model = 0.04 seconds**

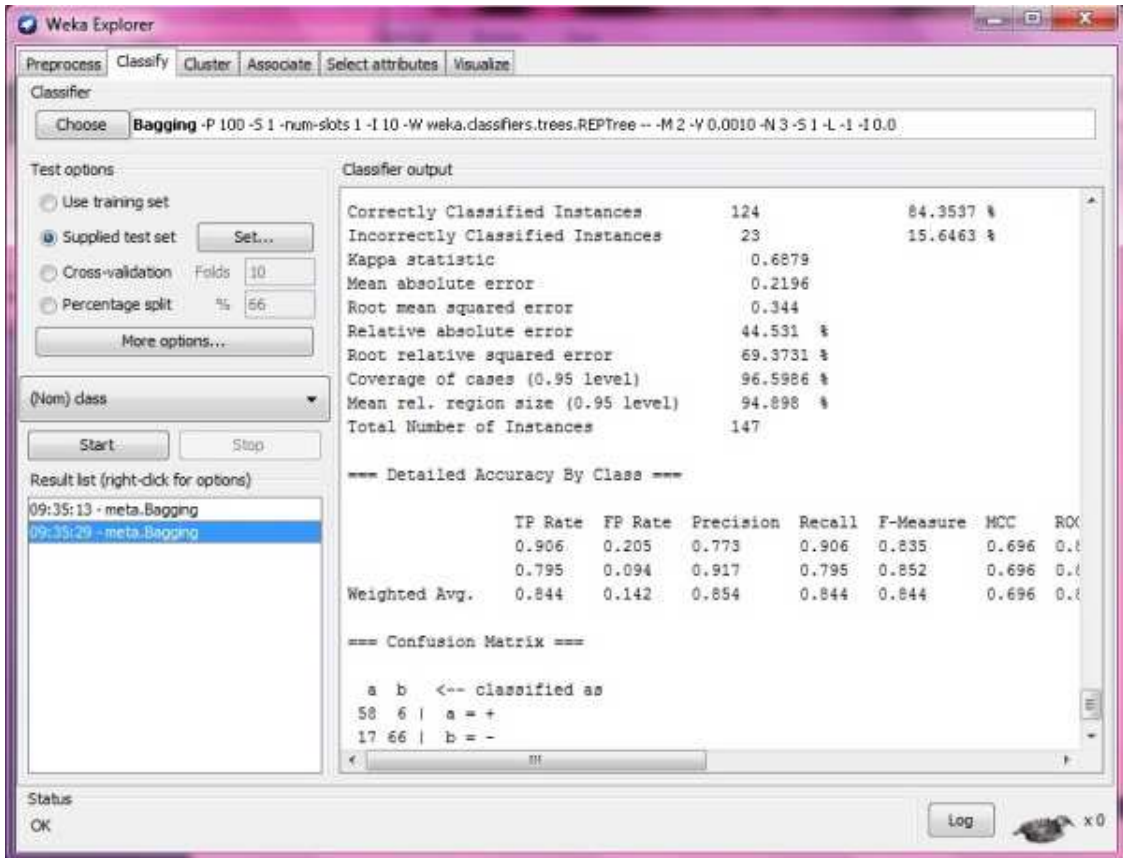


Figure 5.22: Results of Bagging on testingcredit.arff

Results of Bagging:

- Correctly Classified Instances (%) = 84.3537**
- Incorrectly Classified Instances (%) = 15.6463**
- Kappa Statistic = 0.6879**
- Mean Absolute Error = 0.2196**
- F-Measure = 0.835**
- Time Taken to build the Model = 0.05 seconds**

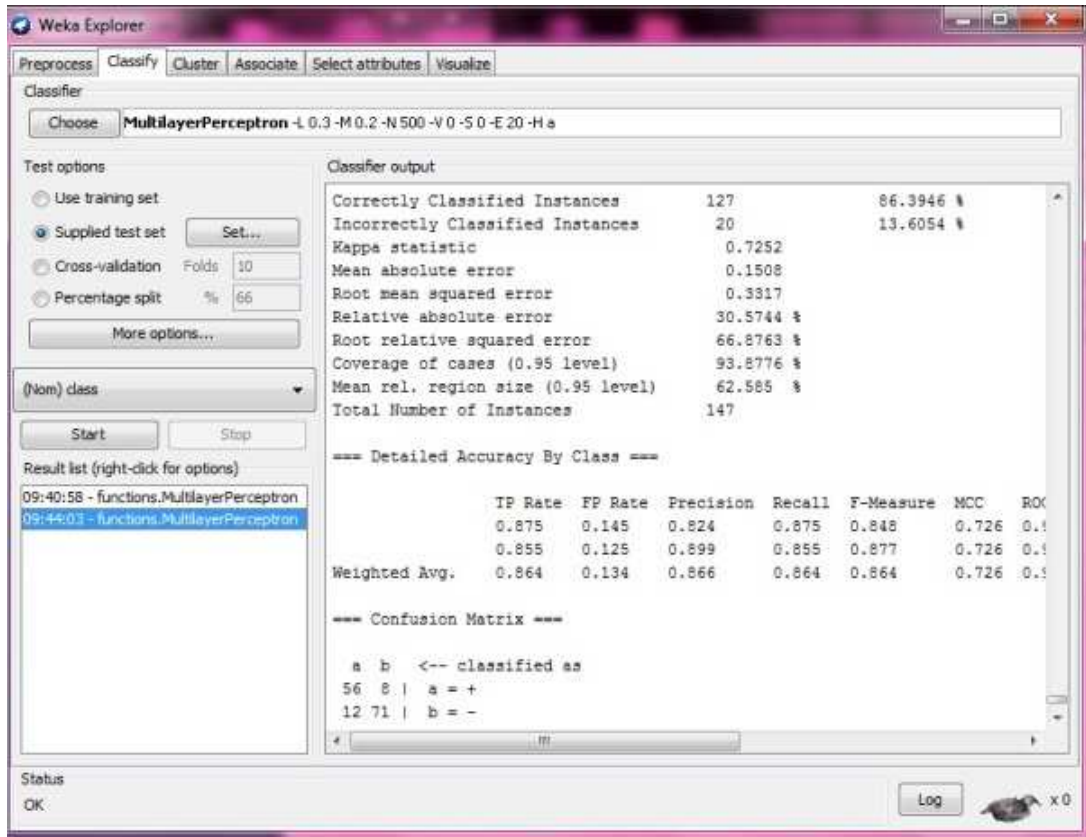


Figure 5.23: Results of MultiLayerPerceptron on testingcredit.arff

Results of MultiLayerPerceptron:

- Correctly Classified Instances (%) = 86.3946**
- Incorrectly Classified Instances (%) = 13.6054**
- Kappa Statistic = 0.7252**
- Mean Absolute Error = 0.1508**
- F-Measure = 0.848**
- Time Taken to build the Model = 0.01 seconds**

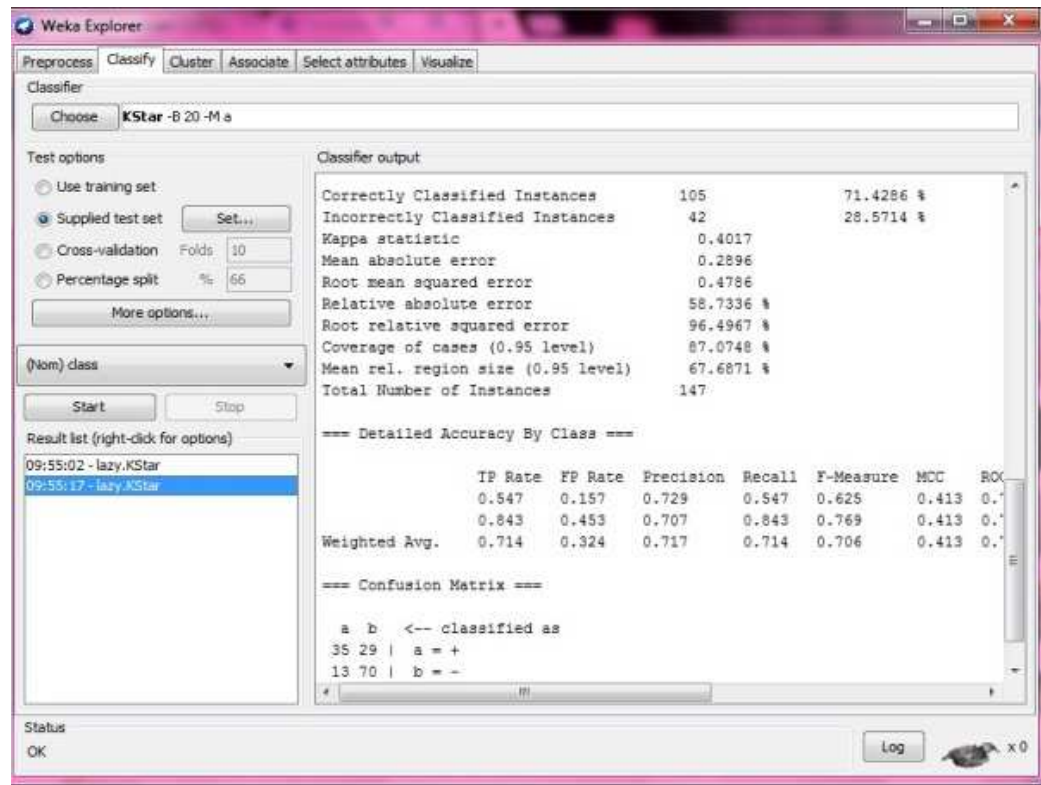


Figure 5.24: Results of KStar on testingcredit.arff

Results of KStar:

- Correctly Classified Instances (%) = 71.4286**
- Incorrectly Classified Instances (%) = 28.5714**
- Kappa Statistic = 0.4017**
- Mean Absolute Error = 0.2896**
- F-Measure = 0.625**
- Time Taken to build the Model = 0 seconds**

5.5. Conclusion

In this section we show the results in the form of charts and tables. Figure 5.25 shows the comparison of all the algorithms with respect to the time taken to build the model.

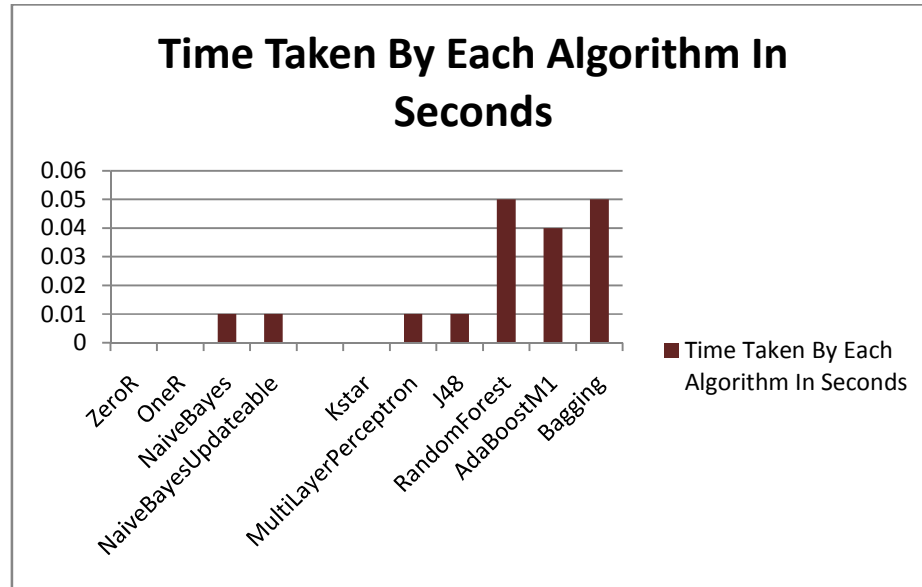


Figure 5.25: Time chart of algorithms

Figure 5.26 shows the comparison based about the number of correctly classified instances by each learning algorithm.

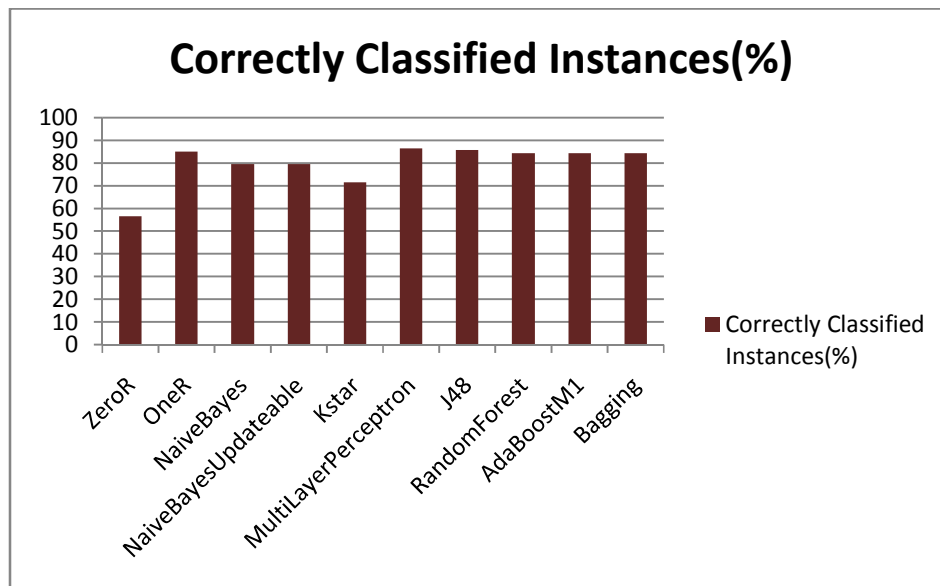


Figure 5.26: Comparison of Algorithms By Percentage Of Correct Instances

In Table 5.4 we have summarized three main measures of evaluation for each algorithm i.e. time taken to build the model, number of correctly classified instances, and F-Measure.

Table 5.4: Comparison of algorithms

Algorithms	Time taken to build model (sec)	Correctly Classified Instances (%)	F-Measure
J48	0.01	85.7143	0.837
RandomForest	0.05	84.3537	0.824
ZeroR	0	56.4626	0
OneR	0	85.034	0.845
NaiveBayes	0.01	79.5918	0.732
NaiveBayesUpdateable	0.01	79.5918	0.732
KStar	0	71.4286	0.625
MultiLayerPerceptron	0.01	86.3946	0.848
AdaBoostM1	0.04	84.3537	0.837
Bagging	0.05	84.3537	0.835

It shows that RandomForest and Bagging take maximum amount of time to build the model i.e.0.05 seconds. Next highest is 0.04 taken by AdaBoostM1. NaiveBayes, NaiveBayesUpdateable and MultiLayerPerceptron take 0.01 seconds and the remaining take 0 seconds to build the model. In terms of second measure of evaluation, MultiLayerPerceptron has the highest percentage of correctly labeled instances and has the best F-Measure among all. Hence, we conclude that MultiLayerPerceptron has performed better than all the other classifiers in the analysis of our dataset.

Chapter 6

Combined Machine Learning and Feature Design

In the previous chapter, we presented an evaluation of the state-of-the-art machine learning algorithms for the task of classification using a real world problem and dataset. We calculated our results on the basis of accuracy of the algorithms in performing classification i.e. predicting the correct output class. In this chapter, we present an approach that shows an increase in the accuracy for solving the classification problems. It is a hybrid approach that combines various learners. We first present a technique of combining learners and also show its implementation using Python programming. Later we discuss feature space design and show its implementation on the combined learner. Section 6.1 provides an introduction for the new concepts used in this chapter that have not been described earlier in this thesis. It provides an idea about the language (Python) we have used for implementing our design, the machine learning tool (Orange) we used for accessing the learning algorithms. Section 6.2 provides an idea about the concept of combining learners, various types of combination techniques and the earlier work done in this regard. In Section 6.3 we discuss the new combined approach, its procedure, experiment and the results. Section 6.4 presents the feature space design, feature selection techniques, steps of feature selection method used, experiment and results.

6.1. Introduction

We first describe some important concepts about Python programming and Orange that we have used in implementing our learning method. In later sections we introduce our new concept and its implementation.

6.1.1. Why Python

These days Python has become a very popular programming/scripting language for the implementation of machine learning concepts. Python is an extensible language. New concepts and functionality is being added continuously in it. Apart from regular programming concepts, it also supports tools for internet e.g. cgi-scripting and xml support. It has a variety of programming tools that makes programming exciting and easier.

Python is a very powerful programming language and is used in a wide variety of application domains. In the area of machine learning it has proved to be very helpful and effective. One of the main reasons of using this language is its intuitive object orientation as OOP paradigm is the most commonly followed paradigm these days. It has full modularity and supports hierarchical

packages. Since our machine learning problems revolve around different types of datasets, we need to be careful about the data types supported by the programming language we use. Python has a very high level dynamic data types. It has a number of extensive standard libraries and third party modules for virtually every task. It can be easily embedded within applications as a scripting interface. More importantly, Python supports portability. We can run the same source code without changing across all implementations. It runs everywhere. It is available for Windows, Linux/Unix, OS/2, Mac, Amiga, and others.

6.1.2. Python Machine Learning tool

Previously we used a machine learning tool WEKA for evaluation which is based on Java. Since we implemented our method in Python, we needed a similar learning tool for Python. There are a number of machine learning tools for Python e.g. PyML (<http://pyml.sourceforge.net/>), MDP (<http://mdp-toolkit.sourceforge.net/>), Shogun (<http://www.shogun-toolbox.org/>), and Orange (<http://orange.biolab.si/>). We used Orange because it supports more classifiers than others and has an interactive graphical user interface. It can also be used for clustering.

Orange is a machine learning tool consisting of functions and objects of C++. This learning tool has a number of machine learning and data mining algorithms and functions for manipulating the data. It is written in C++ and is created for Python. At the user level it is developed using the scripting language Python, which makes it possible for the users to create their algorithms and add them to the existing library. It provides an environment that helps the users to prototype their algorithms faster. It also provides various testing schemes and a number of graphical tools that use functions from library and provide a good user interface. These tools or widgets communicate with each other using signals. These tools can be assembled together to form an application using a graphical environment called Orange Canvas. Widgets can be placed on the canvas and can be connected together to form a schema. Each widget has its own basic function and signals that are passed between these widgets are of different types. Its objects include learners, classifiers, evaluation results, distance matrices, and so forth [Zupan and Demsar, 2008].

Without the use of such machine learning tools, we would have to write the entire code ourselves for all the machine learning tasks e.g. for carrying out cross validation for comparing the

machine learning algorithms, or for loading data and so on. Machine learning toolkits ease the programming by providing in built routines for these tasks thus providing flexibility in experimenting. All we need to do is access these routines from our code.

This machine learning toolkit supports a number of data mining and machine learning tasks ranging from data preprocessing to modeling and evaluation. Some of the techniques supported by this machine learning toolkit are listed below:

- It supports a number of popular data formats e.g. C4.5, Assistant, Retis, and tab-delimited data formats.
- It supports preprocessing and manipulation of data, like sampling of data, scaling and filtering of data, discretization and construction of new attributes, etc.
- It provides support for development of classification models using functions that consist of regression, SVM, classification trees, naive Bayesian classifier.
- It also supports various regression methods i.e. linear regression, regression trees, and instance-based approaches,
- It has support for various wrappers used to calibrate probability predictions of classification models.
- It also supports ensemble approaches.
- It has various association rules and methods used for data clustering.
- It provides various evaluation methods like hold-out schemes and range of scoring methods for prediction models including classification accuracy, AUC, and Brier score. It also supports various hypothesis testing approaches.

The processes on which machine learning algorithms are based are conditional probability estimation, selection and filtering of data, attribute scoring, random sampling, and many others. Orange consists of all these processes in the form of its components that are embedded into algorithms for applying these methods. We can also create new components with the help of Python prototyping and we can use these new components in place of default components or we can use them together with an existing set of components to develop a completely new algorithm. The thing that makes Orange completely different from other machine learning

frameworks is that it supports signal mechanism between different widgets with the help of which they can communicate with each other by exchanging objects.

6.2. Combined Learners

The main reason for combining many learners together is reducing the probability of misclassification due to a single learner by increasing the system's area of knowledge through combination. It is a process of creating a single learning system from a collection of learning algorithms. Learners are combined to achieve a better predictive performance as compared to the performance obtained from individual learners. There are two ways in which learners can be combined together. In the first method, the data is divided into a number of subsets and multiple copies of a single learning algorithm are applied to these different subsets. This method generates multiple hypotheses using the same base learner and follows *variations in data*. In the second method, several learning algorithms are applied to the same application's data. It is a broader concept and such systems are called multiple classifier systems and follow *variation among learners*. As discussed earlier in this work, we cannot have a single learner that suits to all learning problems. For each problem there exists an optimal learning algorithm. By combining the learners we can lessen the risk of choosing a suboptimal learning algorithm by replacing single model selection with model combination.

Our technique of learner combination follows the second method in which several different learners are combined and applied to a single application's data.

6.2.1. Types of Combination Techniques

This section briefly explains different types of techniques for combining the learners and the related literature of these techniques is provided in the next section. Some of the common types of combination techniques are:

- *Bayes optimal classifier*: It is an ideal technique that combines all hypotheses in a hypothesis space. In this technique the hypotheses are given votes based on if a particular hypothesis is true and the training set is sampled from the system. After that the vote given to the hypothesis is multiplied by the initial probability of that hypothesis. The Bayes Optimal Classifier is represented by the following formula:

$$y = \operatorname{argmax}_{c_j \in C} \sum P(c_j | h_i) P(T | h_i) P(h_i), h_i \in H$$

where y denotes the predicted class, C represents the set of all possible classes, H is the hypothesis space, P refers to a probability, and T is the training data.

However, practical implementation of this method is difficult for complex problems. It can be applied only to simple tasks. The reasons for this issue are: the large hypothesis spaces, which are difficult for iteration and determine only a predicted class rather than the probability for each class as required by the term $P(c_j | h_i)$, and its seldom possible to estimate the initial probability for each hypothesis $P(h_i)$.

- *Bootstrap aggregating (bagging) and Boosting:* Both of these methods are based on the *variations in data* method in which the data is divided into a number of subsets and multiple copies of a single learning algorithm are applied to these different subsets. Both these methods combine multiple models built from a single learning algorithm by systematically varying the training data.

Bootstrap aggregating or bagging is a voting method in which each learner in the combined learners votes with equal weight. In this method different training datasets are used to train the base-learners and the training sets are drawn randomly. High accuracy is obtained in the random forest algorithm because random decision trees are combined with bagging in a random forest algorithm [Breiman, 1996]. Voting corresponds to linear combination of learners [Alpaydin, 2010] i.e.

$$y_i = \sum w_j d_{ji} \text{ where } w_j \geq 0, \sum w_j = 1 \quad (1)$$

If A is a learning algorithm and T is a set of training data, the process of bagging takes N samples S_1, \dots, S_N , from T . The algorithm is then applied to each sample independently to make N models h_1, \dots, h_N . When a new query q has to be classified, these models are combined by a simple voting scheme, and the query is assigned a class that has been predicted most often among the N models. Figure 6.1

http://dml.cs.byu.edu/~cgc/docs/ml_dm_tools/Reading/ModelCombination.pdf shows the process of bagging diagrammatically. For generating training datasets, bagging uses bootstrap and the learners are trained using an unstable learning procedure, and an average is taken during testing [Breiman 1996]. This method works effectively if the base learner is unstable i.e. if it is highly sensitive to data i.e. small changes in the training set cause a large difference in the generated learner. This method can be used both for classification

and regression. In case of regression, instead average, median is taken at the time of testing.

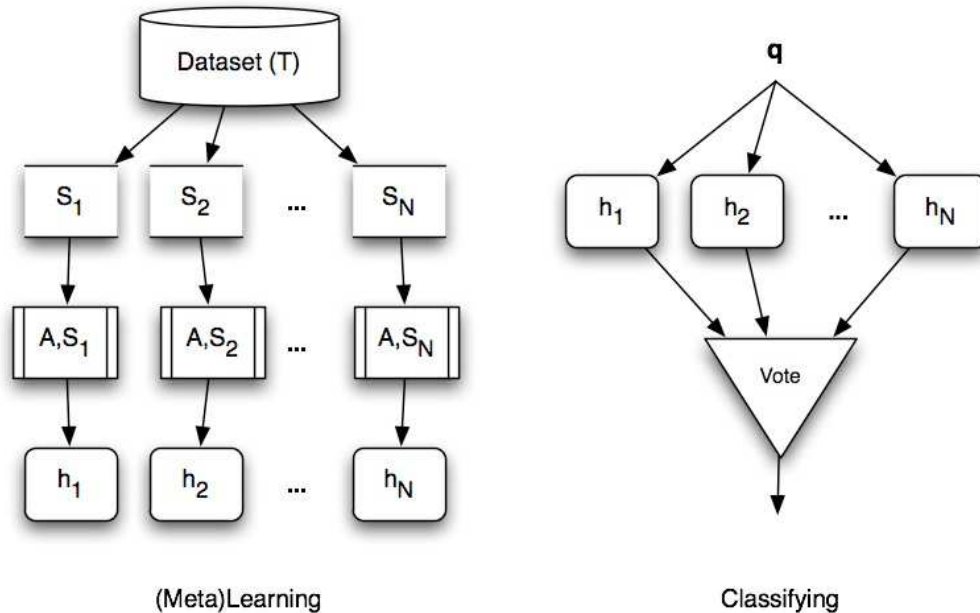


Figure 6.1: Bagging

Boosting [Schapire, 1990] is a process which trains the new instances and combines the learners incrementally in a way such that the focus is laid on the training instances that were previously wrongly classified. In this method the learner is trained on the mistakes of the previous learners. Bagging is based on data variation through a learner's instability and boosting is based on data variation through a learner's weakness. A learner is said to be weak if it derives models that perform slightly better than random guessing. In a weak learner, the error probability is $\frac{1}{2}$. It means for a two-class problem it is better than random guessing and a strong learner has small error probability. The most common example of boosting is adaptive boosting, AdaBoost [Freund and Schapire, 1996]. The process of boosting works by supposing that if a weak learner is run on different distributions repeatedly over the training data, and if the weak classifiers are combined into a single classifier, then it can be made stronger, as illustrated in Figure 6.2 (http://dml.cs.byu.edu/~cgc/docs/ml_dm_tools/Reading/ModelCombination.pdf). The main disadvantage of the boosting method is its need for large training data. AdaBoost does

not suffer this problem as it uses the same training set over and over and thus the training data need not be large, but the classifiers should be simple so that they do not overfit.

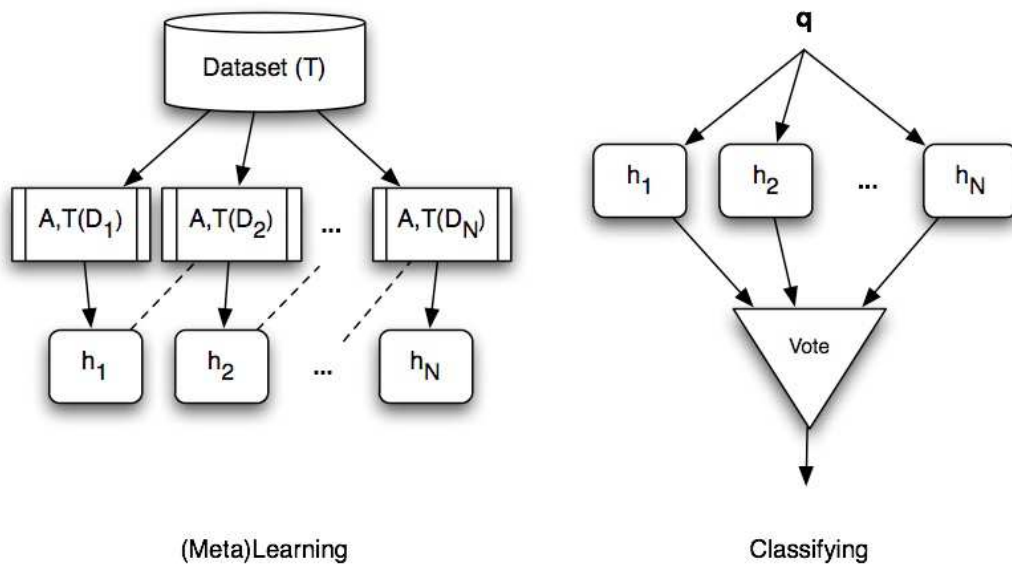


Figure 6.2: Boosting

- Stacking*: This method exploits *variation among learners* in which several learning algorithms are applied to the same application’s data. This method is proposed by Wolpert in 1992. In this method a number of different learning algorithms are run against the dataset which creates a series of models. Then the actual dataset is modified by replacing its each instance by the values that each model predicts for that instance. This creates a new dataset which is given to a new learner that builds the model, as illustrated in Figure 6.3 (http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf). Whenever a new query instance q has to be classified it is first passed through all the learners which create a new query instance q' . Then the model takes it as an input and the final classification for q is produced. For better results it is important in stacked generalization that the learners should be as different as possible so that they will complement each other and these learners should be based on different learning algorithms.

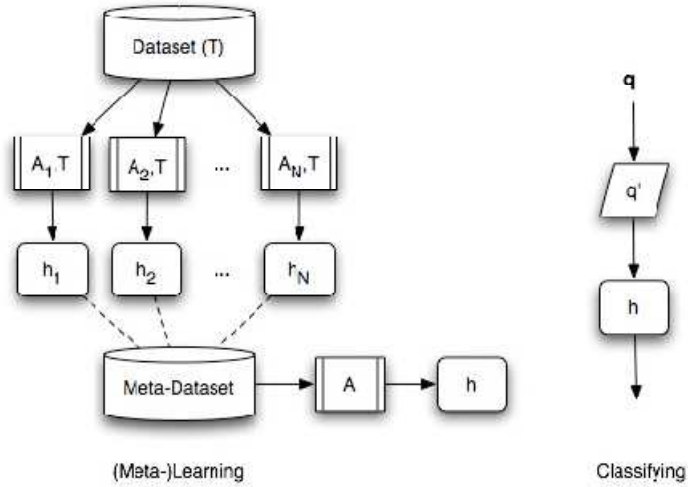


Figure 6.3: Stacking

- Cascading:** This method also follows *variation among learners* approach like stacking but it differs from stacking because stacking uses the learners in parallel whereas cascading uses the learners in sequence. Cascading is a process having multiple stages in which learners are used in sequence i.e. the next learner is used only if the preceding ones are not confident [Alpaydin, 2010]. This method was proposed by Gama and Brazdil. Figure 6.4 (http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf) shows this process.

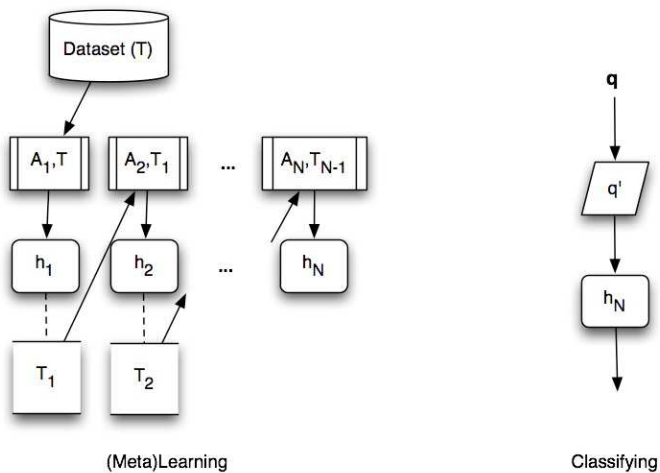


Figure 6.4: Cascading

In cascading the data from the base-level learners is not fed into a single meta-level learner. But each base-level learner also acts as a kind of meta-level learner for the

learner preceding it. The inputs that are fed to the learner consist of the inputs to learner preceding it together with the class probabilities produced by the model induced by the preceding learner. At each step only a single learner is used and the number of steps is unlimited. A new query instance q is converted into a query instance q' by gathering data through the steps of the cascade. The last model of the cascade produces the final classification.

6.2.2. Related Literature

A lot of research work has been carried out in this field. This section presents the work done in the direction of combined learners. A technique called attribute bagging has been developed for improving accuracy and stability of classifier ensembles induced using random subsets of features. This method has been compared with bagging and other methods on a hand-pose recognition dataset and has shown better results than bagging and other methods both in terms of accuracy and stability [Bryll *et al.*, 2002]. Bagging was first introduced by Leo Breiman. He created a method called Bagging Predictors for generating multiple versions of a predictor and used these to create an aggregated predictor [Breiman, 1996]. A Bayesian version of bagging based on the Bayesian bootstrap has been developed. The Bayesian bootstrap has shown to resolve a theoretical problem with ordinary bagging and resulted in more efficient estimators [Clyde and Lee, 2000]. An experimental comparison has been carried out between bagging, boosting and randomization for improving the performance of the decision-tree algorithm C4.5. The experiments have shown that randomization is slightly superior to bagging but not as accurate as boosting in situations with little or no classification noise [Dietterich, 1999]. However, it has been shown that in noisy settings bagging performs much more robustly than boosting. A method of ensemble technique has been developed in which voting methodology of bagging and boosting ensembles has been used with 10 subclassifiers in each one. It has been compared with simple bagging and boosting ensembles with 25 sub-classifiers, and also with other well known combining methods, on standard benchmark datasets and it has been shown that the new is the most accurate [Kotsiantis and Pintelas, 2004]. An algorithm called RankBoost has been developed for combining preferences based on the boosting approach to machine learning. Theoretical results have been shown describing the algorithm's behavior both on the training data, and on new test data not seen during training. Two experiments have been carried

out to assess the performance of RankBoost. In the first experiment, the algorithm has been used to combine different web search strategies, each of which is a query expansion for a given domain. The second experiment has been a collaborative-filtering task for making movie recommendations [Freund *et al.*, 2003]. A statistical perspective on boosting has been proposed with special emphasis on estimating potentially complex parametric or nonparametric models, including generalized linear and additive models as well as regression models for survival analysis. The practical aspects of boosting procedures for fitting statistical models have been illustrated by means of the dedicated open-source software package mboost [Buhlmann and Hothorn, 2007]. Theoretical and practical aspects of boosting and ensemble learning have been discussed and the helpful association that exists between boosting and the theory of optimization has been identified for easing the understanding of boosting [Meir and Ratsch, 2003]. Voting classification algorithms like bagging, boosting and variants have been compared in order to find which of these algorithms use perturbation, reweighting, and combination techniques, and which of the algorithms affect classification error. The authors have shown bias and variance decomposition of the error for showing bias and variance decomposition are influenced by different methods. This comparison has shown that bagging reduces variance of unstable methods, while boosting methods (AdaBoost and Arc-x4) reduce both the bias and variance of unstable methods but increase the variance for Naive-Bayes. It has been found that when probabilistic estimates are used along with no-pruning, then bagging shows an improvement. Mean-squared error of voting methods has been compared to non-voting methods and it has shown that the voting methods show reduction in the errors. They have also examined the problems that arise when boosting algorithms are practically implemented [Bauer and Kohavi, 1998]. Simple online bagging and boosting algorithms have been developed that perform as well as their batch counterparts. Lossless online algorithms for decision trees and Naïve Bayes models have been used [Oza and Russell, 2005]. Cohen has developed stacked sequential learning which is a sequential learning scheme in which an arbitrary base learner is improved so that it becomes aware of the labels of nearby examples. This method has been assessed on various problems. It has been shown that on these problems, the performance of non-sequential base learners improves by sequential stacking; that the performance of learners specially designed for sequential tasks is improved by sequential stacking [Cohen, 2005]. A learning method using multiple stacking for named entity recognition has been proposed which employs stacked

learners using the tags predicted by the lower level learners. This approach has been applied to the CoNLL-2002 shared task to improve a base system [Tsukamoto *et al.*, 2002]. Different methods for interpreting the results of multiple, cascading machine learners have been explored. Each of these methods perform a different task. A framework for modeling cascading learners as a directed acyclic graph has been developed, which has allowed a construction of three-way contingency tables on which various independence tests has been performed. These independence tests have provided insight into how the various learners' performance depends on their predecessor in the chain and/or the inputs themselves [Michelson and Macskassy, 2010]. A technique of localized cascade generalization of weak classifiers has been developed. Using this technique some local regions have been pointed out that have like properties and the cascade generalization of local experts has been used for explaining the relationship between the data characteristics and the target class. This technique has been compared with other well known combining methods using weak classifiers as base-learners, on standard benchmark datasets and it has been shown that this technique is more accurate [Kotsiantis, 2008]. A method has been proposed based on the enrichment of a set of independent labeled datasets by the results of clustering, and a supervised method has been used to evaluate the interest of adding such new information to the datasets. The cascade generalization paradigm has been adapted in the case where an unsupervised and a supervised learner are combined [Candillier *et al.*, 2006]. Bagging, stacking, boosting and error correcting output codes are the main four methods of combining multiple models. These have been discussed covering seven methods of combining multiple learners i.e., voting, bagging, cascading, error-correcting output codes, boosting, mixtures of experts, and stacked generalization [Witten and Frank, 2000]. A theoretical framework for combining classifiers in the two main fusion scenarios has been developed. These two main fusion scenarios are fusion of opinions based on identical and on distinct representations [Kittler, 1998]. For the first scenario i.e. the shared representation they showed that here fusion has been performed with the aim of obtaining a better estimation of the appropriate a posteriori class probabilities. For the second scenario i.e. for the distinct representations it has been pointed out that the techniques based on the benevolent sum-rule fusion are more flexible to errors than those derived from the severe product rule.

6.3. Our approach towards combining learners

In our technique we have used uni-representation approach towards combining learners in which all the learners use the same representation of the input as opposed to multi-representation in which learners use different representations of input data [Alpaydin, 1998]. Combined learners are formed of a number of base learners. The performance of combined learners as a whole is usually much better than that of individual base learners. This process boosts the predicting ability of the learners. Base learners are generated from training data by a base learning algorithm which can be decision tree, neural network or other kinds of machine learning algorithms. As discussed earlier, some methods use a single base learning algorithm to produce homogeneous base learners, but the technique that we follow uses multiple learning algorithms to produce heterogeneous learners.

This section discusses the technique that we use for combining learners. Our technique aims to increase the accuracy of prediction in the classification task. We have used an approach in which multiple learners are combined and class probabilities are computed. We have used our method on a classification task. In case of classification, the class with the highest probability is chosen.

Consider we have to combine N learners (l_1, l_2, \dots, l_N). We represent each learner by l_j and the prediction of each learner l_j by $d_j(x)$. If y represents the final prediction, we can calculate y from the individual predictions of learners, i.e.

$$y = f(d_1, d_2, \dots, d_N | \Phi)$$

f denotes the combining function and Φ represents its parameters [Alpaydin, 2010]. However, for multiple outputs we can get several y 's and we have to choose the class with maximum value for y . In that case, prediction of each learner is represented by $d_{ji}(x)$, $j = 1, \dots, N$, $i = 1, \dots, K$ for K outputs and y_i , $i = 1, \dots, K$ represent the final predictions. For example, in case of classification, we choose the class with the maximum y_i value, i.e.

$$\text{Choose } C_i \text{ if } y_i = \max y_k \text{ where } k = 1 \text{ to } K$$

From equation 1 we get,

$$y_i = \sum w_j d_{ji} \text{ where } w_j \geq 0, \sum w_j = 1$$

In case of classification, the weights approximate to the learner probabilities. Therefore,

$$w_j = P(l_j)$$

$$d_{ji} = P(C_i | x, l_j)$$

The above equation can be rewritten as

$$P(C_i | x) = \sum P(C_i | x, l_j) P(l_j) \text{ for all learners } l_j$$

The class probabilities are calculated using this formula.

6.3.1. Procedure of our approach

In our technique, we take a number of learners and apply them on a single dataset. We designed a technique that takes a number of learners and produces a series of classifiers after applying the learners on the dataset. As far as the task of classification is concerned, it uses all the produced classifiers for calculating the class probabilities and chooses the class for which the classifiers predict the highest probability. Figure 6.5 shows the basic flow of our technique. The steps carried out in our procedure are listed below:

The problem on which we have applied our procedure is a classification problem. In this problem, a function maps the inputs to the desired outputs by determining which of a set of classes a new input belongs to. This is determined on the basis of the training data which contains the instances whose class is known, i.e. $h : X \longrightarrow Y$, where X represents input and Y represents the output class. Let the dataset we use be represented as $D = \{\mathbf{x}_t, y_t\} \ t = 1 \text{ to } T$, where T is used to represent the number of instances in the dataset. Let there be N number of learners that we have to combine i.e. l_1, l_2, \dots, l_N and let K number of output classes in our data i.e. y_t can take values (C_1, C_2, \dots, C_K)

- ❖ For each learner l_j ($j = 1$ to N) in the combination
create the classifier m_j for l_j by training on the dataset D

$$m_j = l_j(D)$$

- ❖ For each class C_i ($i = 1$ to K) in the data
For each classifier m_j ($j = 1$ to N)

Calculate $P(C_i) = P(C_i | \mathbf{x}, m_j)$ that represents the probability that the classifier m_j assigns to the class C_i .

- ❖ Finally, we choose the class with the highest predicted probability, or the class with maximum value for $P(C_i)$ i.e.

Choose C_i if $P(C_i)$ has the maximum value among all $P(C_i)$'s

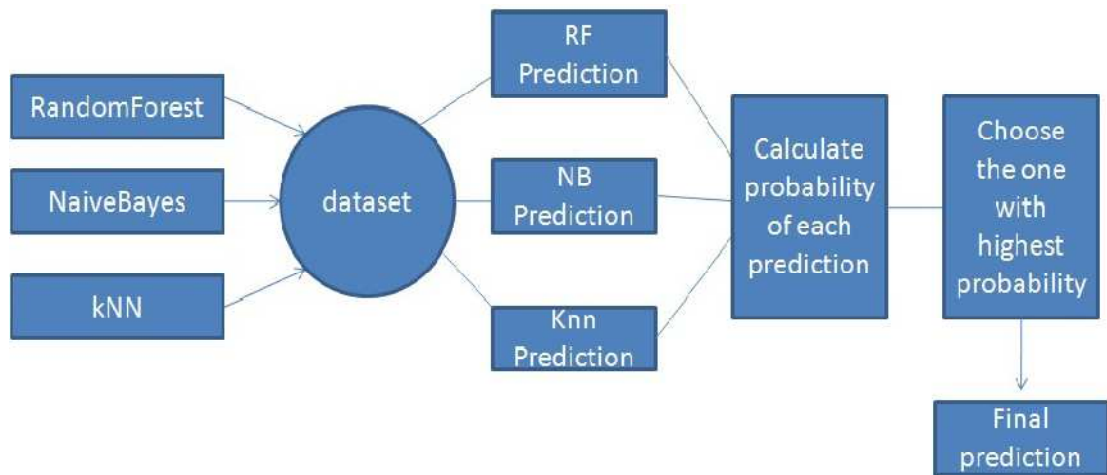


Figure 6.5: Flow of the combined technique

6.3.2. Experimental Setup

As mentioned earlier, for the implementation of the above discussed procedure, we used Python programming and for applying machine learning methods we used Python machine learning tool called Orange. We implemented this approach on the classification problem used in the previous chapter. The dataset (<http://www.hakank.org/weka/credit.arff>) that we used for our experiment for implementing our procedure is the Australian Credit Approval dataset from UCI Repository of Machine Learning Databases and Domain theories (<http://archive.ics.uci.edu/ml/datasets.html>). It is the same dataset that we used in previous chapter for the evaluation of various machine learning algorithms, and its description has already been provided so we skip it here. However, for using the dataset in Orange we had to change its format from ARFF (supported in WEKA) to tab delimited format supported in Orange. The dataset is split into the training and the test sets as done in the previous chapter i.e.

“trainingcredit” and “testingcredit”. The main reason for using the same dataset is to compare the accuracy of the individual learners used in the previous chapter with the accuracy of the combined approach. As discussed earlier, Orange provides a number of inbuilt routines for performing various machine learning tasks. Without its use, we would have to write the entire code ourselves for all the machine learning tasks e.g. for carrying out cross validation for comparing the machine learning algorithms, or for loading data and so on. We provide a list of routines that we used for our approach of combining various learners:

- ❖ First of all, for accessing the learners to be combined we used
`learner = Learner()`
where `Learner()` is a particular learning algorithm in Orange.

- ❖ For loading our dataset in `D`,
`D = orange.Exampletable(“trainingcredit”)`
This loads the dataset that we have used i.e. Credit dataset in `D`.

- ❖ For creating the classifiers by training the learner on the dataset,
`Classifier = learner(D)`
i.e. the learner is called with the data and returns a classifier.

- ❖ For obtaining class probabilities,
`Probabilities = Classifier(D, orange.GetProbabilities)`
Probabilities are stored in a list and using the `max()` routine we find the maximum probability and return the class that has been predicted the highest probability using the `modus()` routine on the list.

- ❖ Finally, for evaluation of our learners, we use cross validation method just as we used in the previous chapter.
`Evaluationresult = orange.crossValidation(learners, D)`

The experiment was carried out in Python machine learning tool. For our experiment we used three learners for combination, i.e. we kept $N = 3$. The algorithms that we used are RandomForest, Naivebayes, and kNN. Then we performed cross validation with 10 folds just like in previous chapter. We split our dataset into training and testing sets.

We carried out our experiment in Python 2.7. It has various modules like IDLE (Python GUI), Python (Command Line), and PythonWin. We used the Script file of PythonWin to develop our application. The file is saved as a script file with “.py” extension. PythonWin has an Interactive Window which allows us to run the commands interactively as well as run our scripts and analyze the results. Figure 6.6 shows loading and running a script file in Interactive Window, and Figure 6.7 shows the results of our script file after it is run.

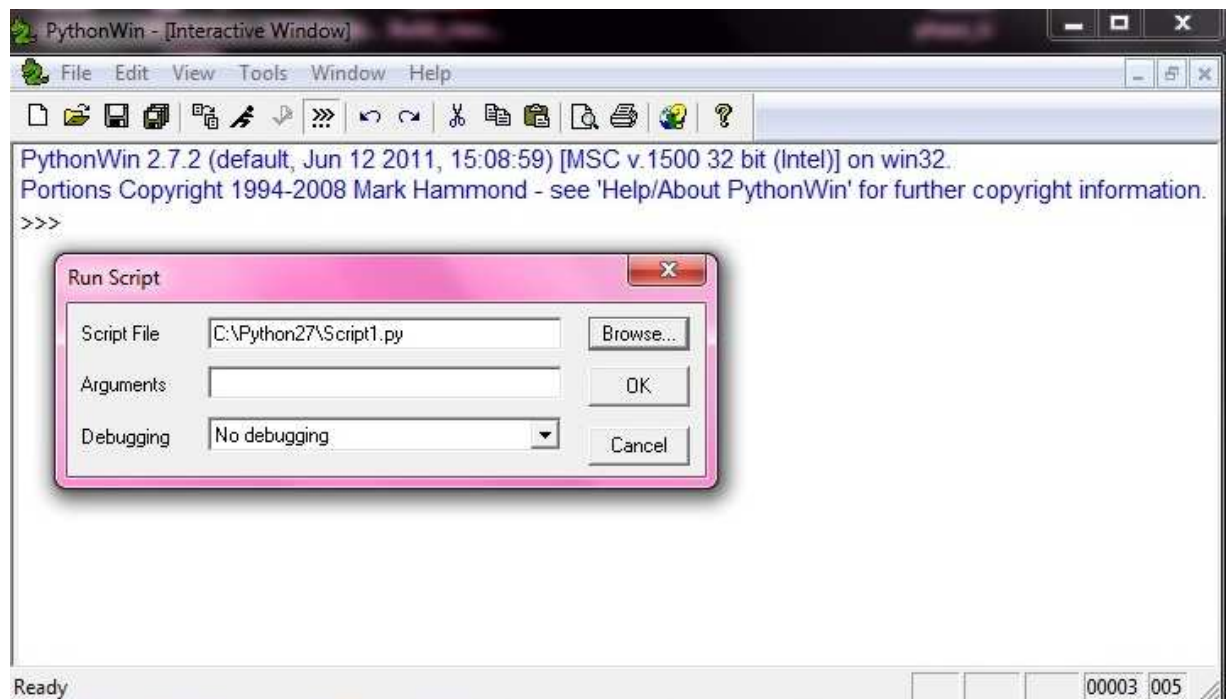


Figure 6.6: Running a Script in Interactive Window in Python

```

PythonWin 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> Results for testingcredit
LEARNERS      ACCURACY  BREIR SCORE  F-MEASURE
RandomForest: 0.845    0.217       0.861
naive bayes:  0.864    0.236       0.881
kNN:          0.831    0.247       0.848
CombinedLearner: 0.870    0.219       0.885

```

Figure 6.7: Results of our script on “testingcredit” file

6.3.3. Results

For evaluating the results of performance comparison of the individual learners and the combined learner, we used F-Measure as used in WEKA in previous chapter. Also we used two additional measures: accuracy and Brier score. We have already discussed Accuracy and F-Measure in Chapter 4.

$$\text{Accuracy} = \frac{tp + tn}{tp + fp + tn + fn}$$

$$\text{Precision}(P) = \frac{tp}{tp + fp}$$

$$\text{Recall}(R) = \frac{tp}{tp + fn}$$

$$\text{F-Measure} = \frac{2 * P * R}{P+R}$$

tp (true positives), *fp* (false positives), *tn* (true negatives) and *fn* (false negatives).

Brier Score: It is a score function that is used to measure the accuracy of probabilistic predictions. It is used in situations where the predictions assign probabilities to a set of outcomes. The outcomes can be binary or categorical in nature. This evaluation measure is proposed by Glenn W. Brier in 1950. It measures the mean squared difference between the predicted probability assigned to the possible outcomes and the actual outcome. Therefore, lower the Brier score, the better the predictions. Table 6.1 shows the comparison of the learners on the basis of accuracy, brier score, and F-Measure.

Table 6.1: Comparison of learners

LEARNERS	ACCURACY	BRIER SCORE	F-MEASURE
RandomForest	0.845	0.217	0.861
NaiveBayes	0.864	0.236	0.881
kNN	0.831	0.247	0.848
Combinedlearner	0.870	0.219	0.885

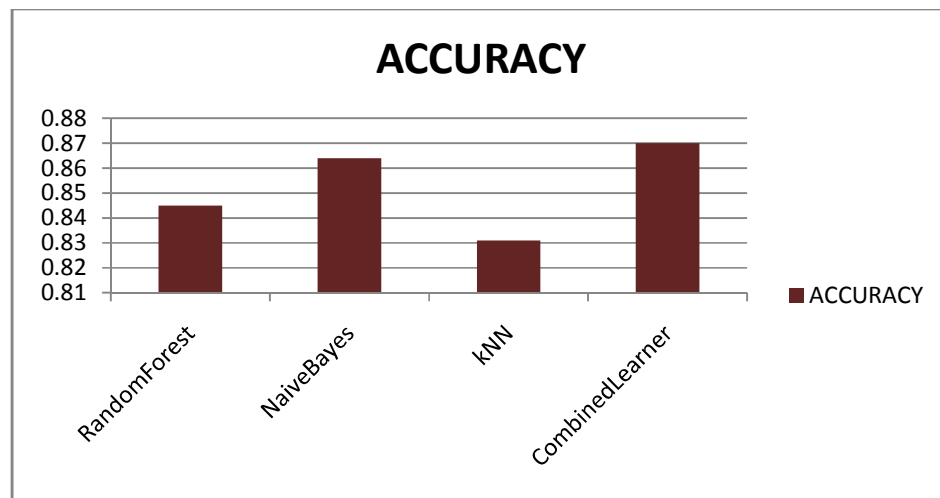


Figure 6.8: Comparison on the basis of Classification Accuracy

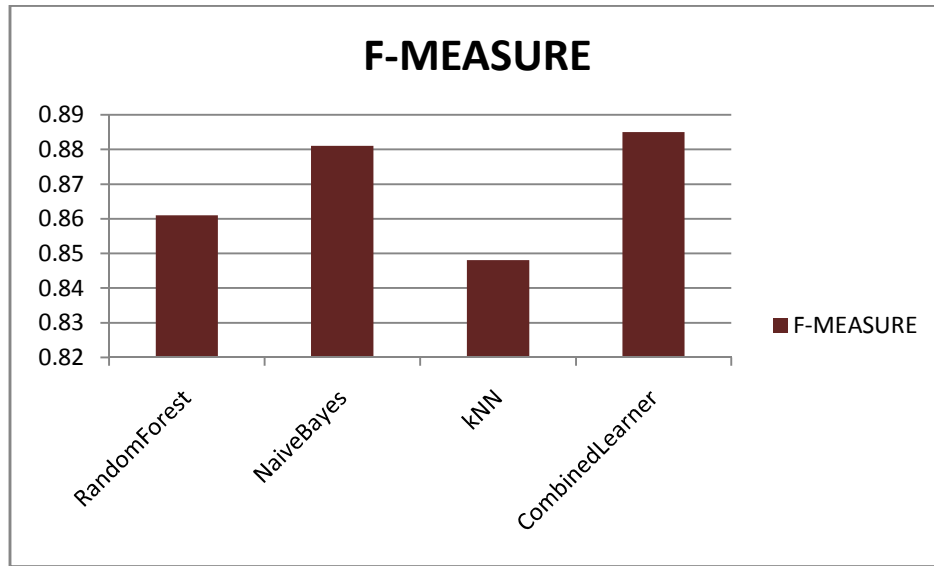


Figure 6.9: Comparison on the basis of F-Measure

Figure 6.8 shows the graphical comparison of various learners on the basis of classification accuracy. It clearly shows that the combined learner has the highest classification accuracy (i.e. 0.870) among all learners. Figure 6.9 shows the graphical comparison of various learners on the basis of F-Measure. It shows that the combined learner has the highest F-Measure (i.e. 0.885). It has highest F-Measure than MultilayerPerceptron (0.848) that was the highest in the evaluation of machine learning algorithms through WEKA in the previous chapter. Therefore, the combined learner outperforms all the learners for our problem of the classification of the credit dataset. Table 6.1 shows that the lowest value (best) for Brier Score is shown by RandomForest (0.217) and the next lowest by our combined approach (0.219).

6.4. Feature Space Design

As discussed in Subsection 3.1.3 of Chapter 3, data preprocessing [Zhang *et al.*, 2002] is an important task of machine learning. Initially the data collected is not directly suitable for training and therefore requires some processing before it can be used for example it may have missing feature values or noise. A number of pre-processing methods have been developed and the decision of deciding which one to use varies according to the situations. If the collected data contains some missing features then a method for handling missing data [Batista & Monard, 2003] is used. Similarly, there are methods for detecting and handling noise [Hodge & Austin, 2004]. Some of the problems with the collected real world data are: data can be incomplete i.e.

some attribute values may be missing, or it may lack certain important attributes, or it may consist of only aggregate data; there can be presence of noise i.e. it may contain errors or outliers; the data may be inconsistent i.e. containing variations in codes or names. Data preprocessing is performed in order to prepare the data for input into machine learning and mining processes. This involves transforming the data for improving its quality and hence the performance of the machine learning algorithms, such as predictive accuracy and reducing the learning time. Once the data preprocessing is complete we get a final training set. A well-known algorithm has been presented for each step of data pre-processing [Kotsiantis *et al.*, 2006].

There are a number of tasks that are carried out in data preprocessing. These are cleaning, normalization, integration, transformation, reduction, feature extraction and selection. Data cleaning involves filling the missing values, smoothing the noisy data, identifying or removing outliers, and resolving inconsistencies. Data integration consists of using multiple databases, data cubes, or files and data transformation involves normalization and aggregation. Data reduction means reducing the volume of the data but producing the same analytical results. Data discretization is part of data reduction which means replacing numerical attributes with nominal ones. Feature extraction and selection are tasks of feature space design. Restructuring the feature space or feature space design is very important and has resulted in a lot of research by the machine learning communities. Researchers have developed several techniques and methods to deal with this problem.

As we have shown before, for our machine learning tasks, data is represented as a table of examples or instances. It is called the dataset. Every instance in the dataset has a fixed number of attributes, or features, along with a label that denotes its class. The features of a dataset contain the information about the problem that we are dealing with and help in the classification process. Usually we believe that if the number of features or attributes is increased in the dataset, it will increase the efficiency of classification. However, by increasing the features there are chances of degradation of the classifier performance [Bishop, 1995]. Usually in many real-world problems, there are a large number of features in the dataset, most of which are irrelevant or redundant. Therefore, an important task in machine learning is deciding and choosing which of the features are relevant and which are irrelevant. Before a classifier can move beyond the training data to make predictions about novel test cases, it must decide which features to use in these predictions

and which to ignore. Therefore it is necessary to find subsets of the feature population that are relevant to the target class and worthy of focused analysis [Blum and Langley, 1997]. This process in which some of the features of the training set are selected and used for classification is called feature selection.

6.4.1. Feature Selection

The most important purpose of feature selection is to make a classifier more efficient by decreasing the size of the dataset. This is necessary for the classifiers that are costly to train e.g. NaiveBayes. The processing time and the cost of the classification systems are increased while their accuracy is decreased if irrelevant and additional features are used in the datasets used for classification. Therefore, it is very important to develop the techniques for selecting smaller feature subsets. However, we have to make sure that the subset which is selected is not so small that the accuracy rates are reduced and the results lack understandability. So it is very important that techniques must be developed that help to find an optimal subset of features from the superset of original features [Witten and Frank, 2000]. There are two ways in which feature selection can be carried out. These are the filter and wrapper approach [Liu and Motoda, 1998]. The filter approach selects a subset of the features that preserves as much as possible the relevant information found in the entire set of features [Kohavi and John, 1997; Freitas, 2002]. Some of the methods that implement filter approach are discussed here. The FOCUS algorithm [Almuallim and Dietterich, 1991] has been designed for noise-free Boolean domains and it follows the MIN-FEATURES bias. It examines all feature subsets and selects the minimal subset of features that is sufficient to predict the class targets for all records in the training set. Another feature selection method that has been developed is called Relief [Kira and Rendell, 1992]. It is an instance-based feature selection method. Relief-F is an extended version of Relief that has been developed for multi-class datasets whereas Relief was designed for two-class problems. In this method an instance is randomly sampled from the data and its nearest neighbor is located from the same and opposite class. The sampled instance is compared to the values of the features of the nearest neighbors and relevance scores for each feature are updated. The process is then carried out repeatedly for many instances. The main idea is that an attribute should be able to differentiate between instances from different classes and should have the same value for instances from the same class. Information gain and gain ratio [Quinlan, 1993] are good examples of measuring the relevance of features for decision tree induction. They use the

entropy measure to rank the features based on the information gained; the higher the gain the better the feature. Moore and Lee [Moore and Lee, 1994] proposed another model using an instance-based algorithm, called RACE, as the induction engine, and leave-one-out cross-validation (LOOCV) as the subset evaluation function. Searching for feature subsets is done using backward and forward hill-climbing techniques. John et al. [John *et al.*, 1994] proposed a similar method and applied it to ID3 and C4.5 on real world domains. Langley et al. [Langley and Sage, 1994] also used LOOCV in a nearest-neighbor algorithm. Caruana et al. [Caruana and Freitag, 1994] test the forward and backward stepwise methods on the Calendar Apprentice domain, using the wrapper model and a variant of ID3 as the induction engine. Wrapper models are usually slower than filter models in the sense that inductive learning is carried out more than once.

6.4.2. Basic Steps in Feature Selection

This section discusses the steps that we followed in selecting the subset of features in our problem. We applied our combined technique on the problem dataset. In Section 6.3 we already evaluated its efficiency. Now we use this method in combination with the feature selection technique. We apply a filter approach on our method that results in a different (filtered) dataset and evaluate the results. The steps that we followed are:

- ❖ Initialize the learner.
 `learner = Learner()`
- ❖ Load the dataset in D,
 `D = orange.Exampletable("trainingcredit")`
 This loads the dataset that we have used i.e. Credit dataset in D.
- ❖ For creating the classifiers by training the learner on the dataset,
 `Classifier = learner(D)`
- ❖ Compute the relevance (R) of the features/attributes. This is done by applying the attribute measure method on the dataset (i.e. `attMeasure(D)`).
- ❖ Set some margin, say m, and remove all those features/attributes for which $R < m$, i.e. whose relevance is below the selected margin. This is done by applying a filter method on the dataset. Only the attributes having $R > m$ are used for classification.
- ❖ Finally, use the learner on both the datasets and compare the accuracy.

6.4.3. Experiment and Results

Again for implementing the above procedure we used Python programming and Python machine learning tool. We carried out the experiment on the same problem and dataset i.e. Credit dataset. Again we use our “testingcredit” file like in previous experiment. Figure 6.10 shows the results of feature subset selection method on “testingcredit” file taking margin 0.010. First it shows the list of all attributes (i.e. 15) in our dataset along with the computed relevance. Then it displays the list of attributes after feature selection process. It displays a reduced list of attributes (i.e. 11). Out of 15 attributes only 11 attributes of our dataset are relevant and the remaining 4 are discarded because their relevance is less than the specified margin (0.010). Finally, it shows the accuracy and the F-Measure of the learners on the dataset after the process of feature selection. Table 6.2 shows the comparison of the performances of the learners based on accuracy and F-Measure with and without feature selection for margin 0.010. The table shows that for all the learners the accuracy and F-Measure either increases or remains same after feature selection. This shows that in our problem only 11 attributes are enough for performing efficiently. Remaining 4 attributes are irrelevant as long as efficiency is concerned. However, we have to take proper care in selecting the margin because the selected subset should not be so small that it reduces the accuracy rates and the understanding of the results. So we need to find an optimal subset of features from the superset of original features.

```

PythonWin - [Interactive Window]
File Edit View Tools Window Help
Before feature subset selection (15 attributes):
0.577 A9
0.119 A6
0.087 A10
0.056 A12
0.048 A1
0.040 A11
0.020 A14
0.013 A4
0.013 A5
0.013 A13
0.010 A8
0.005 A2
0.004 A7
0.003 A3
-0.000 A15

After feature subset selection with margin 0.010 (11 attributes):
0.544 A9
0.115 A6
0.080 A10
0.055 A12
0.048 A1
0.045 A11
0.020 A7
0.019 A14
0.017 A4
0.017 A5
0.012 A8

Results for testingcredit after feature selection
LEARNERS      ACCURACY  BREIR SCORE  F-MEASURE
RandomForest: 0.852    0.224       0.867
naive bayes:  0.864    0.227       0.880
kNN:          0.825    0.242       0.845
CombinedLearner: 0.868    0.226       0.879
Script 'C:\Python27\Script11.py' returned exit code 0
00043 047

```

Figure 6.10: Results of feature subset selection on “testingcredit” with margin 0.010

Table 6.2: Before and after feature selection comparison of learners with margin 0.010

Learners	Accuracy Before feature selection	Accuracy After feature selection	F-Measure Before feature selection	F-Measure After feature selection
RandomForest	0.845	0.852	0.861	0.867
NaiveBayes	0.864	0.864	0.881	0.880
kNN	0.831	0.825	0.848	0.845
CombinedLearner	0.870	0.868	0.885	0.879

Figure 6.11 shows the results of feature subset selection taking margin 0.020. Table 6.3 shows the comparison of the performances of the learners based on accuracy and F-Measure with and without feature selection for margin 0.020. It shows a decrease in the accuracy and F-Measure of

all the learners. After subset selection, only 6 attributes are chosen for classification and remaining attributes are ignored as their relevance is below the margin. But this decreases the overall accuracy of the learners. Hence, for our problem the optimal subset of features is obtained by keeping margin equal to 0.010, which corresponds to 11 out of 15 attributes.

```

PythonWin - [Interactive Window]
File Edit View Tools Window Help
PythonWin 2.7.2 (default, Jun 12 2011, 15:08:59) [MSC v.1500 32 bit (Intel)] on win32
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About.PythonWin' for further copyright information.
>>> Results for testingcredit
Before feature subset selection (15 attributes):
0.577 A9
0.119 A6
0.087 A10
0.056 A12
0.048 A1
0.040 A11
0.020 A14
0.013 A4
0.013 A5
0.013 A13
0.010 A8
0.005 A2
0.004 A7
0.003 A3
-0.000 A15

After feature subset selection with margin 0.020 (6 attributes):
0.508 A9
0.116 A12
0.081 A6
0.081 A1
0.042 A11
0.028 A10

Results for testingcredit after feature selection
LEARNERS    ACCURACY  BREIR SCORE  F-MEASURE
RandomForest: 0.838    0.242    0.854
naive bayes: 0.858    0.233    0.874
kNN: 0.831    0.245    0.843
CombinedLearner: 0.852    0.239    0.869
    
```

Figure 6.11: Results of feature subset selection on “testingcredit” with margin 0.020

In Table 6.4 we have shown the comparison of learners on the basis of their F-Measures without feature selection and with feature selection at two different margins. It is clear that feature selection is important but only as long as it does not decrease the efficiency of the learners by discarding too many attributes on the basis of their relevance. At margin 0.010, learners perform better than without any margin. They show increased or similar efficiency depicting the fact that rest of the attributes were irrelevant. However, at margin 0.020, learners show decrease in performance indicating that too many attributes are being discarded and hence the chosen subset is not an optimal subset.

Table 6.3: Before and after feature selection comparison of learners with margin 0.020

Learners	Accuracy Before feature selection	Accuracy After feature selection	F-Measure Before feature selection	F-Measure After feature selection
RandomForest	0.845	0.838	0.861	0.854
NaiveBayes	0.864	0.858	0.881	0.874
kNN	0.831	0.831	0.848	0.843
CombinedLearner	0.870	0.852	0.885	0.869

Table 6.4: Comparing F-Measure at different margins

Learners	F-Measure Before feature selection	F-Measure After feature selection (margin 0.010)	F-Measure After feature selection (margin 0.020)
RandomForest	0.861	0.867	0.854
NaiveBayes	0.881	0.880	0.874
kNN	0.848	0.845	0.843
CombinedLearner	0.885	0.879	0.869

Chapter 7

Conclusion and Future Work

7.1. Conclusions

These days, machine learning techniques are being widely used to solve real-world problems by storing, manipulating, extracting and retrieving data from large sources. Supervised machine learning techniques have been widely adopted however these techniques prove to be very expensive when the systems are implemented over wide range of data. This is due to the fact that significant amount of effort and cost is involved because of obtaining large labeled data sets. Thus active learning provides a way to reduce the labeling costs by labeling only the most useful instances for learning.

Chapter 2 discusses current developments and applications in NLP and literature survey of various machine learning techniques. We identified the different circumstances in which the learner may ask queries and different querying strategies. Chapter 3 discusses the basic concepts of supervised learning, active learning and learning for complex models. We presented an example of learning pipeline models. We concluded that machine learning strategies that take into consideration the informativeness or the relevance of instances can perform better with fewer labeled examples as compared to other learning approaches. Chapter 4 examines a pipelined approach for information extraction with respect to active machine learning. Machine learning problems solved using a pipeline model show better results. Chapter 5 presents an evaluation of state-of-the-art machine learning algorithms on the basis of efficiency, for the task of classification. Chapter 6 presents a combined approach for the design of a learner that aims at increasing the efficiency of the learning tasks. Machine learning algorithms perform more efficiently for a classification task when they are combined together. For the prediction of the correct output class, combined learner selects the class to which highest probability has been assigned among all the learners. Further we conclude that feature selection is important but only as long as it does not decrease the efficiency of the learners by discarding too many attributes on the basis of their relevance.

7.2. Future Work

The combined approach that we presented in this work has some limitations. Although we have used it on state-of-the-art machine learning algorithms, however, we have evaluated its results on only classification tasks. It can be extended to be used for other important problems e.g.

regression and clustering. Moreover, we theoretically showed how active learning can be applied to part-of-speech tagging and included into the pipeline. In future we intend to show its empirical implementation and performance evaluation using various evaluation metrics. In field of active learning future work involves combining active learning with a subfield of machine learning called transfer learning [Torrey and Shavlik, 2009]. It is applicable in situations when we have a training set available for one problem but not for another similar problem. It involves transferring knowledge from one domain to another to speed up learning.

PUBLICATIONS

- **Towards Understanding Theoretical Developments in Natural Language Processing.** (Mehnaz Khan, Mehraj-ud-Din Dar, S.M.K. Quadri). *International Journal of Computer Applications* (0975 – 8887)Volume 38–No.2, January 2012.
- **Examining a Pipelined Approach for Information Extraction with respect to machine learning.** (Mehnaz Khan, S.M.K. Quadri). *International Journal of Computer Science & Communication Networks*,Volume 2, Number 4, Pages 491-495, ISSN: 2249-5789, August 2012.
- **Evaluating Various Learning Techniques for Efficiency.** (Mehnaz Khan, S.M.K. Quadri). *International Journal of Engineering and Advanced Technology*, Volume-2, Issue-2, Pages 326-331, ISSN: 2249 – 8958, December 2012.
- **An efficient Uni-representation Approach towards Combining Machine Learners.** (Mehnaz Khan, S.M.K. Quadri).IEEE Conference on Information and Communication Technologies ICT 2013. (Accepted for Publication).
- **Effects of using Filter Based Feature Selection on the Performance of Machine Learners using Different Datasets.** (Mehnaz Khan, S.M.K. Quadri). BVICAM's International Journal of Information Technology. (Accepted for Publication).

REFERENCES

- Abe, N. and Mamitsuka, H. Query learning strategies using boosting and bagging. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1–9. Morgan Kaufmann, 1998.
- Abney, S. Bootstrapping. AT&T Laboratories – Research 180 Park Avenue Florham Park, NJ, USA (2002).
- Ahn, L.V. Human Computation. School of Computer Science Carnegie Mellon University Pittsburgh (2005).
- Allwein, E. L., Schapire, R.E. and Singer, Y. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research* **1**, 113–141 (2000).
- Almuallim, H. and Dietterich, T. Learning with Many Irrelevant Features", *Proceedings of the Ninth National Conference on Artificial Intelligence*. pages 547-552. MIT Press, 1991.
- Alpaydin, E. Introduction to Machine Learning. The MIT Press Cambridge, Massachusetts London, England (2010).
- Alpaydin, E. Techniques for Combining Multiple Learners. *Proceedings of Engineering of Intelligent Systems' 98 Conference* (Ed. E Alpaydin), Vol 2, 6-12, ICSC Press, 1998.
- Amershi, S., Fogarty, J., Kapoor, A. and Tan, D. Examining Multiple Potential Models in End-User Interactive Concept Learning. *Machine Learning and Web Interactions*, 1357-1360 (2010).
- Anderson, B. and Moore, A. Active learning for hidden markov models: Objective functions and algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 9-16 (2005).
- Angluin, D. Queries and concept learning. *Machine Learning*, **2**(4):319-342 (1988).
- Angluin, D. Queries revisited. In *Proceedings of the International Conference on Algorithmic Learning Theory*, pages 12–31. Springer-Verlag, (2001).
- Anssi, Y. J., Andras, K., and Jacques, S. Finite-state methods and models in natural language processing. *Natural Language Engineering* **17** (2): 141–144 (2011).

- Arora, S. and Agarwal, S. Active learning for Natural Language Processing. Language Technologies Institute School of Computer Science Carnegie Mellon University. (2007).
- Bach, N. and Badaskar, S. “A Review of Relation Extraction”. Language Technologies Institute, School of Computer Science Canergie Mellon University, Pittsburgh.
- Baldrige, J. and Osborne, M. Active learning and the total cost of annotation. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 9-16 (2004).
- Baram, Y., El-Yaniv, R., and Luz, K. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5:255-291. (2004)
- Batista, G. and Monard, M.C. An Analysis of Four Missing Data Treatment Methods for Supervised Learning, *Applied Artificial Intelligence*, vol. 17, pp.519-533 (2003).
- Bauer, E. and Kohavi, R. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants, *Machine Learning*, vv, 1-38 (1998)
- Baum, E.B. and Lang, K. Query learning can work poorly when a human oracle is used. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, 1992.
- Becker, M. (2008). Active Learning: An Explicit Treatment of Unreliable Parameters. PhD thesis, University of Edinburgh.
- Becker, M. and Osborne, M. A two-stage method for the active learning of statistical grammars. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 991-996 (2005).
- Becker, M., Hachey, B., Alex, B., and Grover, C. Optimising selective sampling for bootstrapping named entity recognition. In *ICML Workshop on Learning with Multiple Views* (2005).
- Bhide, M.A., Gupta, A., Gupta, R., Roy, P., Mohania, M.K. and Ichhaporia, Z. “Liptus: Associating structured and unstructured information in a banking environment”. *Proceedings of the 2007 ACM SIGMOD*, 915-924.
- Bird, S., Klein, E. and Loper, E. 2006. “Natural Language Processing/ Computational Linguistics with Python”.
- Bishop, C. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
- Blitzer, J. *Domain Adaptation of Natural Language Processing Systems*. University of Pennsylvania (2008).

- Blum, A. and Langley, P. Selection of Relevant Features and Examples in Machine Learning. *Artificial Intelligence*, pages 245-271, 1997.
- Blum, A. and Mitchell, T. Combining labeled and unlabeled data with co-training. *COLT: Proceedings of the Workshop on Computational Learning Theory*, Morgan Kaufmann, p. 92-100 (1998).
- Bordes, A., Ertekin, S., Weston, J., and Bottou, L. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579-1619 (2005).
- Boswell, D., 2002. Introduction to Support Vector Machines: University of California, San Diego.
- Brants, T. (2000) TnT - A Statistical Part-of-Speech Tagger, *Proc 6th Applied Natural Language Processing Conference*, ANLP-200.
- Breiman, L. Bagging predictors. *Machine Learning*, **24**(2):123–140, 1996.
- Breiman, Leo (2001). "Random Forests". *Machine Learning* 45 (1): 5–32.
- Brill, E. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics*, December 1995.
- Brinker, K. Active learning of label ranking functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 129-136 (2004).
- Brinker, K. Incorporating diversity in active learning with support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 59-66. AAAI Press, 2003.
- Bryll, R., Osuna, R.G. and Quek, F. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets, *Pattern Recognition* 36 (2003) 1291 – 1302, Elsevier Science Ltd.
- Buhlmann, P. and Hothorn, T. Boosting Algorithms: Regularization, Prediction and Model Fitting, *Statistical Science*, 2007.
- Bunescu, R. C. Learning with probabilistic features for improved pipeline models. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, volume 670-679 (2008).
- Bunescu, R. C., and Mooney, R. J. 2005. "A Shortest Path Dependency Kernel for Relation Extraction". *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, ACL, 724-731.

-
- Campbell, C., Cristianini, N., and Smola, A. Query learning with large margin classifiers. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 111-118 (2000).
 - Candillier, L., Tellier, I., Torre, F. and Bousquet, O. Cascade evaluation of clustering algorithms. (2006).
 - Caruana, R. and Freitag, D. Greedy Attribute Selection. In *Machine Learning: Proceedings of the Eleventh International Conference*, W. Cohen and H. Hirsh (eds). Morgan Kaufmann, 1994.
 - Chan, Y. S. and Ng, H. T. Domain adaptation with active learning for word sense disambiguation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 49-56 (2007).
 - Chang, M.W., Do, Q. and Roth, D. 2006. "Multilingual dependency parsing: A pipeline approach". In *Recent Advances in Natural Language Processing*, 195–204.
 - Chen, J., Schein, A., Ungar, L., and Palmer, M. An empirical study of the behavior of active learning for word sense disambiguation. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, pages 120-127 (2006).
 - ChengXiang, Z. Statistical Language Models for Information Retrieval A Critical Review. *Foundations and Trends in Information Retrieval* 2(3):137–213 (2008).
 - Chin, Y. L. and Eduard, H. From Single to Multi-document Summarization: A Prototype System and its Evaluation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, Philadelphia, (July 2002), 457-464.
 - Claudia, S., Monica, M., and Piek, V. WordNet-LMF: Fleshing out a Standardized Format for WordNet Interoperability. CHI 2009, April 4–9, 2009, Boston, Massachusetts, USA.
 - Cleary, J.G. and L.E. Trigg. 1995. K*: an instance-based learner using an entropic distance measure. In *Proceedings of the 12th ICML-95*, pages 108-114.
 - Clyde, M.A. and Lee, H.K.H. Bagging and the Bayesian Bootstrap, Institute of Statistics & Decision Sciences, Duke University, Durham (2000).

-
- Cohen, W.W. Stacked Sequential Learning, Center for Automated Learning & Discovery School of Computer Science, Carnegie Mellon University, Pittsburgh (2005).
 - Cohn, D., Atlas, L., and Ladner, R. Improving generalization with active learning. *Machine Learning*, **15**(2):201-222 (1994).
 - Cohn, D., Ghahramani, Z. and Jordan, M.I. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, (1996).
 - Collins, M. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 1-8 (2002).
 - Cooper, W.S. The inadequacy of probability of usefulness as a ranking criterion for retrieval system output. University of California, Berkeley (1971).
 - Cortes, C. and Vapnik, V.N. Support-vector networks. *Machine Learning*, **20**(3):273–297, 1995.
 - Cristian, D.N.M., Michael, G., and Susan, D. Mark My Words! Linguistic Style Accommodation in Social Media In *Proceedings of WWW 2011*, Hyderabad, India. ACM, 1 April 2011.
 - Culotta, A. and McCallum, A. Reducing labeling effort for structured prediction tasks. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 746-751 (2005).
 - Culotta, A., Krisjansson, T., McCallum, A., and Viola, P. Corrective feedback and persistent learning for information extraction. *Artificial Intelligence*, **170**(14):1101-1122 (2006).
 - Dagan, I. and S. Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 150- 157. Morgan Kaufmann, (1995).
 - Dasgupta, S., Hsu, D. and Monteleoni, C. A general agnostic active learning algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, volume 20, pages 353–360. MIT Press, (2008).

-
- David, M. Z., Bonnie, J. D., and Jimmy, L. Single-Document and Multi-Document Summarization Techniques for Email Threads Using Sentence Compression. College of Information Studies, University of Maryland 2008.
 - Day, D., Aberdeen, J., Hirschmann, L., Kozierok, R., Robinson, P. and Vilain, M. Mixed-initiative development of language processing systems. In *Fifth Conference on Applied Natural Language Processing ANLP-97*, pages 348–355, New Brunswick, NJ. Association for Computational Linguistics (1997).
 - Dietterich, T.G. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, *Machine Learning*, 1-22 (1999).
 - Dimitrakakis, C. and Savu-Krohn, C. Cost-minimizing strategies for data labeling: Optimal stopping and active learning. In *Proceeding of the International Symposia on Foundations of Information and Knowledge Systems (FoIKS)*, pages 96-111 (2008).
 - Ding, Z., Jiang, B., Shuyi, Z., Hongyuan, Z., and Lee, G. Exploring Social Annotations for Information Retrieval. ACM, Beijing, China (2008).
 - Dipanjan, D., and Andre, F. T. M. A Survey on Automatic Text Summarization. Language Technologies Institute, Carnegie Mellon University (2007).
 - Donmez, P. and Carbonell, J. Optimizing estimated loss reduction for active sampling in rank learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 248-255 (2008).
 - Donmez, P., Carbonell, J. G., and Bennett, P. N. Dual strategy active learning. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 116-127 (2007).
 - Dragomir, R. R., Kathleen, M., and Eduard, H. Introduction to special issue on summarization. *Association for Computational Linguistics*, 28(4), 399-408 (2002).
 - Dredze, M. and Crammer, K. Active learning with confidence. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 233-236 (2008).

-
- Druck, G., Settles, B., and McCallum, A. Active learning by labeling features. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 81-90 (2009).
 - Dubois, V. and Quafafou, M. "Concept learning with approximation: Rough version spaces". *Rough Sets and Current Trends in Computing: Proceedings of the Third International Conference, RSCTC 2002*. Malvern, Pennsylvania. pp. 239–246.
 - Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification*. Wiley-Interscience, 2nd edition, (2001).
 - Edmundson, H. P. 1968. *Mathematical Models in Linguistics and Language Processing*.
 - Ellen, M. V. Overview of the TREC. Question Answering Track. National Institute of Standards and Technology, Gaithersburg, MD 20899 (2001).
 - Ellen, M. V. Overview of the TREC. Question Answering Track. National Institute of Standards and Technology, Gaithersburg, MD 20899 (2002).
 - Elworthy, D. Question answering using a large NLP system. The Ninth Text Retrieval Conference (TREC 9) (2000).
 - Federov, V. *Theory of Optimal Experiments*. Academic Press, 1972.
 - Fine, S., Gilad-Bachrach, R., and Shamir, E. Query by committee, linear separation and random walks. *Theoretical Computer Science*, **284**:25-51 (2002).
 - Finkel, J. R.; Manning, C. D.; and Ng, A. Y. 2006. "Solving the problem of cascading errors: Approximate Bayesian inference for linguistic annotation pipelines". In *Proc. Of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
 - Finn, A. and Kushmerick, N. Active learning selection strategies for information extraction. In *Proceedings of the International Workshop on Adaptive Text Extraction and Mining*, pages 18-25 (2003).
 - Franc, V., Zien, A. and Scholkopf, B. Support vector machines as probabilistic models. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, 665–672.

- Francis, B., Hitoshi, I., Sanae, F., Kiyotaka, U., Takayuki, K., and Kyoko, K. 2009. Enhancing the Japanese WordNet. Proceedings of the 7th Workshop on Asian Language Resources, ACL-IJCNLP 2009, 1-8.
- Freitas, A. Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag, 2002.
- Freund, Y. and Schapire, R. E. An decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, **55**(1):119-139 (1997).
- Freund, Y., Iyer, R., Schapire, R.E. and Singer, Y. An Efficient Boosting Algorithm for Combining Preferences, *Journal of Machine Learning Research* 4 (2003) 933-969.
- Freund, Y., Seung, H.S., Shamir, E and Tishby, N. Selective sampling using the query by committee algorithm. Machine Learning, **28**:133–168, 1997.
- Freund, Y. and Schapire, R. Experiments with a new Boosting Algorithm. In Machine Learning: Proceedings of the Thirteenth International Conference, 148-156. (1996).
- Fujii, A., Tokunaga, T., Inui, K. and Tanaka, H. Selective sampling for example-based word sense disambiguation. Computational Linguistics, **24**(4):573–597, 1998.
- Geman, S., Bienenstock, E., and Doursat, R. Neural networks and the bias/variance dilemma. Neural Computation, 4:1-58 (1992).
- Geoffrey, Z., and Patrick, N. 2009. A Segmental CRF Approach to Large Vocabulary Continuous Speech Recognition. Microsoft Research, Redmond, WA.
- Gerard, M., and Gerhard, W. Towards a Universal WordNet by Learning from Combined Evidence. CIKM'09, November 2–6, 2009, Hong Kong, China.
- Gerasimos, P., Chalapathy, N., Guillaume, G., Ashutosh, G., and Andrew, W. S. 2003. Recent Advances in the Automatic Recognition of Audio-Visual Speech. In Proceedings of the IEEE, 91(9), September 2003.
- Ghahramani, Z. Unsupervised learning. In O. Bousquet, G. Raetsch, & U. von Luxburg (Eds.), *Advanced lectures on machine learning*. Berlin: Springer-Verlag (2004).
- Gregory, D., Gideon, S. and Andrew, M. Learning from labeled features using generalized expectation criteria., SIGIR, 595-602. ACM (2007).

-
- Gunn, S.R. “MATLAB Support Vector Machines.” University of Southampton, Electronics and Computer Science, URL <http://www.isis.ecs.soton.ac.uk/resources/svminfo/> (1998).
 - Guo, Y. and Greiner, R. (2007). Optimistic active learning using mutual information. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 823-829.
 - Guo, Y. and Schuurmans, D. Discriminative batch mode active learning. In *Advances in Neural Information Processing Systems (NIPS)*, number 20, pages 593–600. MIT Press, Cambridge, MA, 2008.
 - Haffari, G. and Sarkar, A. Active learning for multilingual statistical machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 181-189 (2009).
 - Haffari, G., Roy, M., and Sarkar, A. Active learning for statistical phrase-based machine translation. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, pages 415-423 (2009).
 - Hand, David J. (1998): “Reject inference in credit operations,” in *Credit Risk Modeling: Design and Application* (ed. E. Mays), 181-190, AMACOM.
 - Hauptmann, A., Lin, W., Yan, R., Yang, J. and Chen, M.Y. Extreme video retrieval: joint maximization of human and computer performance. In *Proceedings of the ACM Workshop on Multimedia Image Retrieval*, pages 385–394. ACM Press, 2006.
 - Haussler, D. Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1):7–40, 1994.
 - Henning, W., Benno, S., and Gregor, E. “Constructing Efficient Information Extraction Pipelines”. CIKM’11 ACM, Scotland, UK (2011).
 - Herbrich, R., Graepel, T. and Williamson, R.C. The Structure of Version Space, Microsoft Research Ltd., Cambridge, U.K. National ICT Australia, Canberra, Australia (2004).
 - Hinton, G. E. and Sejnowski, T. J. Unsupervised Learning: *Foundations of Neural Computation*, MIT Press, Cambridge, MA, MIT Press Publishers, 1999.

- Hiroya, T., and Manabu, O. Text Summarization Model based on Maximum Coverage Problem and its Variant. *Proceedings of the 12th Conference of the European Chapter of the ACL*, 781–789, Athens, Greece, 30 March – 3 April 2009.
- Hodge, V. and Austin, J. A Survey of Outlier Detection Methodologies, *Artificial Intelligence Review*, Volume 22, Issue 2, pp. 85-126 (2004).
- Hoi, S.C.H., Jin, R. and Lyu, M.R. Large-scale text categorization by batch mode active learning. In *Proceedings of the International Conference on the World Wide Web*, pages 633–642. ACM Press, 2006a.
- Hoi, S.C.H., Jin, R., Zhu, J. and Lyu, M.R. Batch mode active learning and its application to medical image classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 417–424. ACM Press, 2006b.
- Hwa, R. Sample selection for statistical parsing. *Computational Linguistics*, **30**(3):73–77, 2004.
- Japkowicz, N. and Stephen, S. The Class Imbalance Problem: A Systematic Study *Intelligent Data Analysis*, Volume 6, Number 5, (2002).
- Jiang, J. A literature survey on domain adaptation of statistical classifiers. Tech report (2008).
- Joachims, T. Making large scale SVM learning practical. *Advances in Kernel Methods-Support Vector Learning*. MIT-Press
- John, G., Kohavi, R. and Pfleger, K. Irrelevant Features and Subset Selection Problem. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 121-129. Morgan Kaufmann Publishers, 1994.
- Jones, R. Learning to Extract Entities from Labeled and Unlabeled Text. PhD thesis, Carnegie Mellon University (2005).
- Jones, R., Ghani, R., Mitchell, T., and Riloff, E. Active learning for information extraction with multiple view feature sets. In *Proceedings of the ECML Workshop on Adaptive Text Extraction and Mining (ATEM)* (2003).
- Jun, W., and Jianhan, Z. Portfolio Theory of Information Retrieval. *SIGIR'09*, July 19–23, Boston, Massachusetts, USA (2009).

-
- Jurafsky, D. and Martin, J. H. *Speech and Language Processing*. Prentice Hall, 2nd edition (2008).
 - Jurafsky, D. and Martin, J.H. *Speech and language processing: an introduction to natural language processing, computational linguistics and speech recognition*. Upper Saddle River, NJ: Prentice Hall (2000).
 - K.H. Ng, *Commercial Banking in Singapore*. Singapore: Addison Wesley, 1996, pp. 252-253.
 - Kambhatla, N. 2004. “Combining Lexical, Syntactic, and Semantic Features with Maximum Entropy Models for Extracting Relations”. *Proceedings of the ACL 2004*.
 - Kapoor, A., Horvitz, E. and Basu, S. Selective supervision: Guiding supervised learning with decision-theoretic active learning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 877–882. AAAI Press, 2007.
 - Keller, F. (2002) *Evaluation Connectionist and Statistical Language Processing*.
 - Kevin, M., Michael, B., Jules, B., Wendy, C., John, G., Dilip, G., James, H., and Elizabeth, L. “Implementation and Evaluation of a Negation Tagger in a Pipeline-based System for Information Extraction from Pathology Reports”. *MEDINFO*, 663-667 (2004).
 - Khan, M. and Quadri, S.M.K. (2012a). “Examining a Pipelined Approach for Information Extraction with respect to machine learning”. *International Journal of Computer Science and Communication Networks*, **Vol 2**(4):491-495.
 - Khan, M. and Quadri, S.M.K. (2012b). “Evaluating Various Learning Techniques for Efficiency”. *International Journal of Engineering and Advanced Technology (IJEAT)*, **Vol 2**(2):326-331.
 - Khan, M., Dar, M. and Quadri, S.M.K. (2012). “Towards Understanding Theoretical Developments in Natural Language Processing”. *International Journal of Computer Applications*, **Vol 38**(2):1-5.
 - Kim, S., Song, Y., Kim, K., Cha, J.-W., and Lee, G. G. Mmr-based active machine learning for bio named entity recognition. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics (NAACL)*, pages 69-72 (2006).

- King, R. D., Whelan, K. E., Jones, F. M., Reiser, P. G., Bryant, C. H., Muggleton, S. H., Kell, D. B., and Oliver, S. G. (2004). Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, **427**(6971):247-252.
- Kira, K. and Rendell, L. The Feature Selection Problem: Traditional Method and a New Algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 129-134. MIT Press, 1992.
- Kittler, J. On Combining Classifiers. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, VOL. 20, NO. 3, MARCH 1998.
- Kohavi, R. and John, G. Wrappers for Feature Subset Selection. *Artificial Intelligence'97*, pages 273-324, 1997.
- Kornai, A. *Extended Finite State Models of Language (Studies in Natural Language Processing)*, Cambridge University Press (1999)
- Kosmopoulos, A., Paliouras, G., and Androutsopoulos, I. Adaptive spam filtering using only naïve bayes text classifiers. In *Proceedings of the Conference on Email and Anti-Spam (2008)*.
- Kotsiantis, S. Locally application of cascade generalization for classification problems. *Intelligent Decision Technologies* **2** (2008) 1-8.
- Kotsiantis S.B. Supervised machine learning: a review of classification techniques. *Informatica* 31:249–268, 2007.
- Kotsiantis, S. B. and Pintelas, P. E. Combining Bagging and Boosting, *INTERNATIONAL JOURNAL OF COMPUTATIONAL INTELLIGENCE VOLUME 1 NUMBER 4* 2004.
- Kotsiantis, S., Kanellopoulos, D. and Pintelas, P. "Data Preprocessing for Supervised Learning", *International Journal of Computer Science*, 2006, Vol 1 N. 2, pp 111-117.
- Krishnamurthy, V. Algorithms for optimal scheduling and management of hidden markov model sensors. *IEEE Transactions on Signal Processing*, **50**(6):1382–1397, (2002).
- Kristan, M., Skocaj, D. and Leonardis, A. Online Kernel Density Estimation For Interactive Learning. *Journal of Image and Vision Computing* (2009).

-
- Krysta, M. S., Lucy, V., and Christopher, J.C. Enhancing Single-document Summarization by Combining RankNet and Third-party Sources. Association for Computational Linguistics. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, 448–457, Prague, June 2007.
 - Krzysztof, G. and Daniel, S. Preference elicitation for interface optimization. *In Proceedings of UIST 2005*.
 - Kumaran, A., Naren, D., Ashok, B., Saravanan, K., Anil, A., Ashwani, S., Sridhar, V., Vidya, N., Vikram, D., and Sandor, M. WikiBABEL: A System for Multilingual Wikipedia Content, in in Proceedings of the 'Collaborative Translation: technology, crowd sourcing, and the translator perspective' Workshop (co-located with AMTA 2010 Conference), Denver, Colorado, Association for Machine Translation in the Americas, 31 October 2010.
 - Kumarana, Narend, Ashwani, S., and Vikram, D. WikiBhasha: Our Experiences with Multilingual Content Creation Tool for Wikipedia, in Proceedings of Wikipedia Conference India, Wikimedia Foundation (2011).
 - Kupiec, J. 1992. Robust part-of-speech tagging using a hidden markov model. *Computer Speech and Language*, **6**:225-242.
 - Lafferty, J., McCallum, A. and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 282–289. Morgan Kaufmann, 2001.
 - Langley, P. and Sage, S. Oblivious Decision Trees and Abstract Cases. In Working Notes of the AAAI94 Workshop on CaseBased Reasoning, 1994.
 - Lars, B., and Markus, F. All in the Family: A Comparison of SALDO and WordNet. Sprakbanken, University of Gothenburg, Sweden (2009).
 - Laws, F. and Schutze, H. Stopping criteria for active learning of named entity recognition. In *Proceedings the International Conference on Computational Linguistics (COLING)*, pages 465-472 (2008).

- Lewis, D. and Gale, W. A sequential algorithm for training text classifiers. In Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval, pages 3–12. ACM/Springer, (1994).
- Lewis, D. D. and Catlett, J. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 148-156 (1994).
- Liang, P., Jordan, M. I., and Klein, D. Learning from measurements in exponential families. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 641-648 (2009).
- Liddy, E., Diamond, T., and McKenna, M. DR-LINK in TIPSTER III. *Information Retrieval*, 3, 291-311 (2000).
- Liere, R. and Tadepalli, P. (1997). Active learning with committees for text categorization. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 591-597.
- Lindenbaum, M., Markovitch, S. and Rusakov, D. Selective sampling for nearest neighbor classifiers. *Machine Learning*, **54**(2):125–152, 2004.
- Liu, H. and Motoda, H. *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, Boston/Dordrecht/London, Boston, 1998.
- Liu, Y. Active learning with support vector machine applied to gene expression data for cancer classification. *Journal of Chemical Information and Computer Sciences*, 44:1936–1941, 2004.
- Liu, Y. *New Issues in Credit Scoring Applications* (2001).
- Lodhi, H., Saunders, C., Taylor, J.S., Cristianini, N. and Watkins, C. 2002. “Text Classification Using String Kernels”. *Journal of Machine Learning Research*, 419-444.
- Manning, C. and Schütze, H. *Foundations of Statistical Natural Language Processing*. MIT Press (1999).
- Manuel, C. D. G., Maria, T. M. V., Alfonso, U. L., and Jose, M. P. O. Using WordNet in Multimedia Information Retrieval. Workshop. Springer-Verlag Berlin Heidelberg. 185-188 (2011).

- Marine, C., and Dekai, W. Improving Statistical Machine Translation using Word Sense Disambiguation. Department of Computer Science and Engineering, University of Science and Technology, Clear Water Bay, Hong Kong, 61-72 (2007).
- Markovitch, S. and Rosenstein, D. Feature Generation Using General Construction Functions, *Machine Learning* **49**: 59-98 (2002).
- Martin, C., Michael, G., and Joel, T. The utility of article and preposition error correction systems for English language learners: Feedback and assessment, in *Language Testing*, Sage, July 2010.
- McCallum, A. and Nigam, A. A comparison of event models for naive bayes text classification. In *Proceedings of the AAAI Workshop on Learning for Text Categorization*, pages 41–48, 1998a.
- McCallum, A. and Nigam, K. Employing EM in pool-based active learning for text classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 359–367. Morgan Kaufmann, 1998b.
- Meir, R. and Ratsch, G. *An Introduction to Boosting and Leveraging*. (2003).
- Melville, P. and Mooney, R. Diverse ensembles for active learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 584–591. Morgan Kaufmann, 2004.
- Michael, A. C., Remco, V., Masataka, G., Marc, L., Christophe, R., and Malcolm, S. Content-Based Music Information Retrieval: Current Directions and Future Challenges. *Proceedings of the IEEE* 96(4):668-696, April 2008.
- Michel, G., and Chris, Q. Optimal Search for Minimum Error Rate Training, in *Proc. of Empirical Methods in Natural Language Processing*, July 2011.
- Michelson, M. and Knoblock, C. “Semantic annotation of unstructured and ungrammatical text”. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 1091–1098 (2005).
- Michelson, M. and Macskassy, S.A. Judging the Performance of Cascading Models: A First Look, Fetch Technologies, 841 Apollo St, Ste. 400, El Segundo, CA 90245 USA (2010).

-
- Michie, D., Spiegelhalter, D.J., Taylor, C.C. and Campbell, J. Machine learning, neural and statistical classification. Ellis Horwood, Upper Saddle River, NJ, USA, 1994. ISBN 0-13-106360-X. Data available at <http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/>
 - Mitchell, T. Generalization as search. *Artificial Intelligence*, **18**:203–226, (1982).
 - Mitchell, T. *Machine Learning*. McGraw-Hill, 1997.
 - Moore, W. and Lee, S. Efficient Algorithms for Minimizing Cross Validation Error. In *Machine Learning: Proceedings of the Eleventh International Conference*, 1994.
 - Murphy, K.P. (2006) *Naïve Bayes Classifiers*.
 - Ngai, G. and Yarowsky, D. Rule writing or annotation: Cost-efficient resource usage for base noun phrase chunking. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 117-125 (2000).
 - Nguyen, H. T. and Smeulders, A. Active learning using pre-clustering. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 623-630 (2004).
 - Nicholas, J. B. 2008. Some(what) Grand Challenges for Information Retrieval. *ACM SIGIR FORUM*. 42(1):47-54 June 2008.
 - Nilsson, N.J. *Introduction to Machine Learning*, Department of Computer Science Stanford University, 2005.
 - Nowozin, S. and Lampert, C.H. *Structured Support Vector Machines*. Microsoft Research, IST Austria, 2011.
 - Oza, N.C. and Russell, S. *Online Bagging and Boosting*. Computer Science Division, University of California, Berkeley, 2005.
 - Patrick, P., and Ariel, F. Jigs and Lures: Associating Web Queries with Strongly-Typed Entities, in *Proceedings of Association for Computational Linguistics - Human Language Technology (ACL-HLT-11)*, June 2011.
 - Pearl, P. and Chen, L. User-involved preference elicitation for product search and recommender systems. *AI Magazine*, **29**(4) (2009).
 - Pelosof, R., Jones, M. and Ying, Z. Speeding-up margin based learning via stochastic curtailment. In *ICML/COLT Budgeted Learning Workshop*, Haifa, Israel, June 25 2010.

-
- Provost, F., Fawcett, T. and Kohavi, R. The Case Against Accuracy Estimation for Comparing Induction Algorithms. In Shavlik, J. (Ed.), *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 445-453 San Francisco, CA. Morgan Kaufmann.
 - Quinlan, J. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, 1993.
 - Rabiner, L. R. A tutorial on hidden markov models and selected applications in speech recognition IEEE, **77**(2):257-286 (1989).
 - Radev, D. R., Hovy, E., and McKeown, K. Introduction to the special issue on summarization. *Computational Linguistics*. 28(4):399-408. [1, 2] (2002).
 - Ratnaparkhi, A. A Maximum Entropy Model for Part-of-Speech Tagging, University of Pennsylvania Dept. of Computer and Information Science, 1996.
 - Raymond, J. M. 2007. Learning for Semantic Parsing. *Computational Linguistics and Intelligent Text Processing: Proceedings of the 8th International Conference, CICLing 2007*, Springer, Berlin, Germany, 311-324.
 - Roche, E., and Shabes, Y. *Finite-State Language Processing (Language, Speech and Communication)*, MIT Press (1997).
 - Roth, D. and Small, K. Active learning for pipeline models. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 683-688 (2008).
 - Roth, D. and Small, K. Margin-based active learning for structured output spaces. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 413-424 (2006b).
 - Roy, N. and McCallum, A. (2001). Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 441-448.
 - Russell, S. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition (2003).
 - Sarawagi, S. (2007). Information Extraction. *Foundations and Trends in Databases*, 261-377.

- Sarawagi, S. and Bhamidipaty, A. 2002. “Interactive Deduplication using Active Learning”. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD-2002)*, Edmonton, Canada.
- Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning* **5**(2), 197–227
- Scheffer, T. and Wrobel, S. Active learning of partially hidden markov models. In *Proceedings of the ECML/PKDD Workshop on Instance Selection* (2001).
- Scheffer, T., Decomain, C., and Wrobel, S. Active hidden markov models for information extraction. In *Proceedings of the International Conference on Advances in Intelligent Data Analysis*, pages 309-318 (2001).
- Schein, A. I. and Ungar, L. H. Active learning for logistic regression: An evaluation. *Machine Learning*, **68**(3):235-265 (2007).
- Schohn, G. and Cohn, D. Less is more: Active learning with support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 839-846 (2000).
- Settles, B. “Active Learning Literature Survey”, Computer Sciences Technical Report 1648, University of Wisconsin–Madison (2010).
- Settles, B. 2009. “Active Learning. Advanced Statistical Language Processing”. Machine Learning Department, Carnegie Mellon University.
- Settles, B. and Craven, M. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1069–1078. ACL Press, 2008.
- Seung, H. S., Opper, M., and Sompolinsky, H. Query by committee. In *Proceedings of the Annual ACM Workshop on Computational Learning Theory (COLT)*, pages 287-294 (1992).
- Shannon, C.E. A mathematical theory of communication. *Bell System Technical Journal*, **27**:379–423,623–656, 1948.
- Shen, D., Zhang, J., Su, J., Zhou, G., and Tan, C.-L. Multi-criteria-based active learning for named entity recognition. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 589-596 (2004).

-
- Silber, H.G., and McCoy, K. F. Efficient text summarization using lexical chains In: H. Lieberman(Ed.). *Proceedings of IUI 2000 International Conference on Intelligent User Interfaces*, 9-12 Jan. 2000, New Orleans, LA. New York: ACM. 252-255.
 - Sindhvani, V., Melville, P., and Lawrence, R. D. Uncertainty sampling and transductive experimental design for active dual supervision. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 953-960 (2009).
 - Song, J. and Zhao, D.Y. Study of automatic abstracting based on corpus and hierarchical dictionary. *Journal of Software*,11, 308-14 (2000).
 - Steinwender, J. and Bitzer, S. Multilayer Perceptrons, A discussion of *The Algebraic Mind* 2003, University of Osnabrueck, (2003).
 - Stock, O. Natural language processing and intelligent interfaces. *Annals of Mathematics and Artificial Intelligence*, 28, 39-41 (2000).
 - Surdeanu, M., Moldovan, D.I. and Harabagiu, S.M. Performance Analysis of a Distributed Question/Answering System. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 13(6):579-596 (2000).
 - Tang, M., Luo, X., and Roukos, S. Active learning for statistical natural language parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 120-127(2002).
 - Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin markov networks. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*.
 - Thompson, C.A., Califf, M.E. and Mooney, R.J. Active learning for natural language parsing and information extraction. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 406–414. Morgan Kaufmann, 1999.
 - Tim, P., Michael, G., Scott, C., David, M. C., and Aman, D. Predicting the Importance of Newsfeed Posts and Social Network Friends, *American Association for Artificial Intelligence* , July 2010.
 - Tomanek, K., Wermter, J., and Hahn, U. (2007). An approach to text corpus construction which cuts annotation costs and maintains reusability of annotated data. In *Proceedings*

- of the *Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 486-495.
- Tong, S. and Chang, E. Support vector machine active learning for image retrieval. In *Proceedings of the ACM International Conference on Multimedia*, pages 107–118. ACM Press, 2001.
 - Tong, S. and Koller, D. Support vector machine active learning with applications to text classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 999–1006. Morgan Kaufmann, 2001.
 - Toutanova, K. and Manning, C. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *EMNLP/VLC 1999*, pages 63–71.
 - Toutanova, K., Klein, D., Manning, C. and Singer, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *NAACL*.
 - Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 823-830.
 - Tsukamoto, K., Mitsuishi, Y. and Sassano, M. Learning with Multiple Stacking for Named Entity Recognition, Fujitsu Laboratories (2002).
 - Tur, G., Hakkani-Tur, D. and Schapire, R.E. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, **45**(2):171–186, 2005.
 - Turmo, J., Ageno, A. and Catala, N. Adaptive Information Extraction, *ACM Computing Surveys*, **38**(2) (2006).
 - Van den Bosch, A., Daelemans, W. and Weijters, A. 1996. Morphological analysis as classification: an inductive-learning approach. In K. Ofiazer and H. Somers, editors, *Proceedings of the Second International Conference on New Methods in Natural Language Processing*.
 - Vlachos, A. A stopping criterion for active learning. *Computer Speech and Language*, **22**(3):295-312 (2008).
 - Vlachos, A. Active annotation. In *Proceedings of the Workshop on Adaptive Extration and Mining*, pages 64-71 (2006).

- Voorhees, E. The TREC-8 question answering track report. [Online] Available: <http://trec.nist.gov/pubs/trec8/papers/qa-report.pdf> (1999).
- Voorhees, E. The TREC-9 question answering track report. [Online] Available: <http://trec.nist.gov/pubs/trec9/papers/qa-report.pdf> (2000).
- Voutilainen, A. (1995). Morphological disambiguation. In (Karlsson et al.,1995), chapter 6, p. 165–284.
- Waldrop, M.M (2001). Natural language processing, *Technology Review*, 104, 107-108.
- Watanabe, K., Bollegala, D., Matsuo, Y. and Ishizuka, M. “A Two-Step Approach to Extracting Attributes for People on the Web”. ACM, Madrid, Spain (2009).
- Witten, I. and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques with JAVA Implementations*. Morgan Kaufmann Publishers, 2000.
- Xu, Z., Akella, R., and Zhang, Y. Incorporating diversity and density in active learning for relevance feedback. In *Proceedings of the European Conference on Information Retrieval (ECIR)*, pages 246-257 (2007).
- Yan, R., Yang, J. and Hauptmann, A. Automatically labeling video data using multi-class active learning. In *Proceedings of the International Conference on Computer Vision*, pages 516–523. IEEE Press, 2003.
- Yan, R., Yang, J., and Hauptmann, A. Automatically labeling video data using multiclass active learning. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 516-523 (2003).
- Yu, H. SVM selective sampling for ranking with application to data retrieval. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 354–363. ACM Press, (2005).
- Yu, L. and Liu, H. Efficient Feature Selection via Analysis of Relevance and Redundancy, *JMLR*, 5(Oct):1205-1224 (2004).
- Yu, X. and Lam, W. Bidirectional Integration of Pipeline Models. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)* (2010).
- Zhang, C. and Chen, T. An active learning framework for content based information retrieval. *IEEE Transactions on Multimedia*, 4(2):260–268, 2002.

-
- Zhang, S., Zhang, C., Yang, Q. Data Preparation for Data Mining. Applied Artificial Intelligence, Volume 17, pp. 375-381 (2002).
 - Zhang, S.X. and Gales, M.J.F. Structured Support Vector Machines for Noise Robust Continuous Speech Recognition, Department of Engineering, University of Cambridge, Cambridge, UK (2011).
 - Zhang, T. and Oles, F. J. A probability analysis on the value of unlabeled data for classification problems. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1191-1198 (2000).
 - Zhao, S. and Grishman, R. 2005. "Extracting relations with integrated information using kernel methods". *Proceedings of the 43rd Annual Meeting On Association for Computational Linguistics*, 419-426.
 - Zhu, J. and Hovy, E. H. Active learning for word sense disambiguation with methods for addressing the class imbalance problem. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 783-790 (2007).
 - Zhu, J., Wang, H., and Hovy, E. H. Learning a stopping criterion for active learning for word sense disambiguation and text classification. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*, pages 366-372 (2008a).
 - Zhu, J., Wang, H., and Hovy, E. H. Multi-criteria-based strategy to stop active learning for data annotation. In *Proceedings the International Conference on Computational Linguistics (COLING)*, pages 1129-1136 (2008b).
 - Zhu, X. Semi-supervised learning literature survey. Computer Sciences, University of Wisconsin-Madison (2008).
 - Zhu, X., Lafferty, J., and Ghahramani, Z. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML Workshop on the Continuum from Labeled to Unlabeled Data*, pages 58-65 (2003).
 - Zupan, B. and Demsar, J. Open-Source Tools for Data Mining, Clin Lab Med 28 (2008) 37-54, Elsevier Inc.