8-2005

Towards a scalable and efficient data classification technique.

Fadi Samih Omar Mehanna
University of Louisville

TOWARDS A SCALABLE AND EFFICIENT DATA CLASSIFICATION TECHNIQUE

By

Fadi Samih Omar Mehanna
B.A., Cairo, Egypt, 1997

A Thesis
Submitted to the Faculty of the
Graduate School of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science in Computer Science

Computer Engineering and Computer Science Department
Speed Scientific School
University of Louisville
Louisville, Kentucky

August, 2005

TOWARDS A SCALABLE AND EFFICIENT DATA CLASSIFICATION TECHNIQUE

By

Fadi Samih Omar Mehanna
B.A., Cairo, Egypt, 1997

A Thesis Approved on

May 31, 2005

by the following Thesis Committee:

____
Prof. Dr. Adel Elmaghraby
Thesis Director

____
Dr. Khaled Wahba

____
Dr. Ahmed Emam

DEDICATION

This thesis is dedicated to my precious mother

Mrs. Seham El-Sayed Shaykhoun

who has given me valuable and unceasing support.

I love you so much mother.

ACKNOWLEDGMENTS

I would like to thank my major professor, Dr. Adel El-Maghraby, for his guidance and patience. He was very helpful in solving complex and critical difficulties during the whole period of my study. Dr. Khaled Wahba is the one who has facilitated and solved any technical problem I have faced. He has helped me during the hard times of my study in a very friendly way, for that I would like to give him my special thanks. I would also like to thank Dr. Ahmed Emam for his patience and the friendly environment he has made while guiding me until I have submitted this thesis. Dr. Emam was such a friend before being a professor. He made me feel like dealing with my close friend. Dr. Emam, Thank you for such a wonderful relationship. I would also like to express my thanks to my colleagues and my other professors for their understanding and patience during the critical times of completing this thesis. They guided me truly and honestly and put me on the right path.

ABSTRACT

TOWARDS A SCALABLE AND EFFICIENT DATA CLASSIFICATION TECHNIQUE

Fadi Samih Omar Mehanna

June 2, 2005

Data Classification is a task that could be found in many life activities. In general, the term could be used for any activity that derives some decision or forecast based on the currently available information. Using a more accurate definition, a classification procedure is the construction of some kind of a method for making judgments for a continuing sequence of cases, where each new case must be assigned to one of pre-defined classes. This type of construction has been termed supervised learning, in order to distinguish it from unsupervised learning or clustering in which the classes are not pre-defined but are concluded from the available data.

This thesis is divided into five chapters, analyzing three classification techniques, namely nearest neighbor technique, perceptron learning algorithm and multi-layer perceptrons with backpropagation, based on performance and scalability issues. Chapter one gives an introduction to the research topic of this thesis. In addition it states the problem that builds the core of this thesis and predefines the objective of this study, namely selecting the most efficient and scalable classification algorithm that suits a given classification task. Chapter two explores a historical review of the literature introduced in

the classification domain. It focuses mainly on the topics that are related to this study and presents some of the new classification approaches. Chapter three introduces the way based on which this thesis is designed. The technical methodology used to analyze and investigate the three classification algorithms is clearly described. In this thesis different experiments are introduced to prove the findings. The datasets used here are considered to be real-life datasets that present sports players and cars classification tasks.

Chapters four and five represent the main core of this thesis, as they contain the data analysis, main findings and conclusions that are derived from different experiments. The nearest neighbor classification technique is one of the lazy learners because before the classification process starts, it needs to store all of the training samples. But, although it takes more time to classify any unknown samples, it is considered the most efficient technique among other classification techniques. A natural and future step would be using the single-layer perceptron algorithm that does not need to store the data samples to reach an acceptable convergence rate. Alternatively, it speeds the recognition or the learning process, because it learns and stores only the weights of the neural network used to implement the algorithm. This algorithm has a big deficiency: it only works for the linearly separable data samples. So, it is now a suitable phase to start working on a more scalable and efficient technique. It is the multi-layer perceptrons network with backpropagation that has the power of solving different complex and non-linearly separable classification tasks.

TABLE OF CONTENTS

PAGE

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

CHAPTER I

INTRODUCTION

Background

Information is the backbone of today's businesses. Different kinds of data, such as client databases, inventory databases, medical and statistical databases, support the evolving demands of today's economy. As the size of data increases rapidly from day to day, organizations must have powerful and efficient plans to manage the data. Understanding the value of our data is considered the core point of development in businesses and in scientific researches.

Databases are full of hidden and rich information that can be used for enhancing the business decisions. Classification is a type of data analysis that can be used to extract functions describing important data classes. Data classification is the process of finding a set of models that distinguish and differentiate data classes, for the purpose of predicting the class of given data samples whose class label is unknown or for the purpose of checking if the pre-classified data samples have been already classified into the correct classes. The resulting model is based totally on the analysis of the training dataset that contains data samples whose class label is already known. The derived model could be represented in many forms, such as neural networks, mathematical formulas or decision trees.

Data classification is a process that is divided into two steps. In the first step, a model is derived describing predetermined set of data classes. The model is built by analyzing the training dataset records which in turn are described by individual attributes or features. Each record is assumed to belong to a predefined class; it is determined by the class label attribute. The samples in the training dataset are randomly selected from the object space. Because the class label of each training sample is already known and provided in the dataset, this step is also known as supervised learning. In the case of the class label is not provided, the step could be known as unsupervised learning or clustering. After building the model in the first step, the second step takes place. In the second step, the model is then used to start the actual classification process. This process is applied on the testing dataset which contains the remaining samples from the object space. For each test sample, the given class label which represents the desired output is compared with the learned output or the actual output for that sample.

Different classification methods have been introduced by many researchers in various fields, such as machine learning, expert systems, statistics and medicine. Most classification and learning algorithms are memory resident, as they assuming a small data size. But recently more advanced techniques, that are considered scalable to handle large resident data, have been widely proposed. As the recent classification problems are getting more complex and are containing very large data, those new techniques play a very important role in building scalable and efficient classification systems.

Problem Definition

An important component of many data analysis techniques is finding a good and efficient classification algorithm; this process requires a very careful planning in order to maximize the chances of success. Different criteria interfere deeply in finding which classification algorithm helps in solving the given classification problem. Size of data repositories, complexity of the given problems and, period of time needed to complete the learning process are some of those important criteria. Each of these classification algorithms may be suitable for a number of certain problems; but not all of these algorithms may be used to solve any classification problem.

Intensive Investigating of pitfalls that may occur through implementing classification algorithms is a bottleneck in today's researches. This is critical because data classification is particularly useful when a certain data repository contains hidden information that can be used for future decision making; e.g.; for medical diagnosis, for scientific data analysis, or for statistical analysis. Researches have a vast range of various classification algorithms at their disposal, including nearest neighbor technique, decision tree induction, perceptron learning algorithm and error backpropagation. Over the part years, many updates of these algorithms have been introduced. These updates were targeting increasing the effectiveness of these algorithms. So, today's researches confront a major problem in using those algorithms, namely how one could choose the best algorithm to be implemented for solving a new classification problem in an efficient way.

Thesis Objectives

This thesis addresses the different criteria that help in selecting the best algorithm through practical and experimental analysis of some of the most famous and widely used algorithms. It also introduces a performance evaluation for each one of the used algorithms to help in deciding the most efficient algorithm for a specific classification problem. The algorithms used here are: nearest neighbor technique, single-layer perceptron and multi-layer perceptron with backpropagation. The objective here is to progress gradually through three phases, where each algorithm is implemented separately in a separate phase, from the less efficient algorithm to the most powerful and scalable one based on different criteria, such as complexity of the classification problem and process or learning time. In each phase, the logic of implementing the algorithm is examined and investigated extensively to reach an accurate decision about its performance.

At the end of each algorithm analysis, one could then decide if the algorithm could solve a given classification problem or not. If it could, there is no need to step forward and investigate the second algorithm. But if it couldn't, one could then start investigating the next algorithm and so on. In some cases the nearest neighbor method will be the most efficient technique used to solve the problem, but in other cases a more efficient and scalable algorithm could be then used. Knowing which algorithm is suitable to a certain classification problem and setting up plans to tackle different types of classification problems are considered main objectives of this study.

Thesis Methodology

The methodology for this thesis is based on investigating and analyzing three different classification algorithms, namely nearest neighbor method, single-layer perceptron and multi-layer perceptron with error backpropagation. The architecture of each one of these algorithms is examined intensively to build a solid base for the future process of choosing the most appropriate algorithm which would be used to solve some given classification problem.

The experimental part of each algorithm gives a practical and applied experience that helps totally in achieving the objectives of this thesis. MATLAB is the tool that has been used widely to analyze and implement each algorithm. The logic behind each one of the implemented algorithms is coded into MATLAB M-files. In the phase of single-layer perceptron and the phase of MLP with error backpropagation manual computations have been tackled before starting the code implementation in MATLAB environment.

All of the experimental examples introduced in this thesis depend on previous prepared datasets that have been carefully selected from some of the famous object spaces. The dataset used for the nearest neighbor method contains information about the height and weights of four types of sports players. For the single-layer perceptron, the dataset used is a subset of the sports players' dataset but it contains only two types of sports players. The last and most important dataset is the one used for implementing the MLP with error backpropagation. The dataset describes five classes of cars. It contains 14 features or attributes that differentiate between the five classes.

Thesis Structure

The thesis is structured in five chapters, namely introduction, literature review, research design and methodology, data analysis and findings and conclusion and future work. The first chapter contains general information about the research topic and defines the technical problem for which this thesis has been created. The second chapter describes and introduces some of the most valuable researches that have been implemented in data classification techniques. In addition, it shows the most important results and findings of those studies. The third chapter presents the general methodology which is used to design the research. It describes also the datasets used to implement the experiments.

The fourth chapter is considered to be the section of data analysis and findings which is the core of this thesis as it goes through the main topic of the research and offers some detailed findings which will play a very important role in confronting the problem that has been defined above. . This chapter presents and examines the three algorithms on which this thesis depends. The fifth chapter states the final conclusions that have been derived totally from the findings described in the previous chapter.

CHAPTER II

LITERATURE REVIEW

This chapter introduces some of the previous valuable efforts and researches that have significantly contributed towards the success of data classification field in computer sciences. As the data classifications techniques have greatly evolved in the past few decades, many industries and businesses have taken the first step in using data classification widely to analyze their data and to grab knowledge and valuable hidden information from their exiting databases. The data classification is now spreading in all life branches, so that some of the most important problems arising in commerce, industry and science can be considered as classification or even decision problems using complex and large data.

Previous Studies in Classification

D. Michie, D.J. Spiegelhalter and C.C. Taylor (1994) as editors have introduced in their book "Machine Learning, Neural and Statistical Classification" different approaches that have been considered and main historical approaches of research in classification field, namely statistical, machine learning and neural network. They have shown how all the researches were attempting to derive some kind of classifiers that would be able to equal a human decision-maker's behavior, but in the same time are consistent, to handle a wide range of classification problems and to be used in real life situations with proven success. In the book J.M.O. Mitchell has addressed the first approach, namely statistical approach. He identified two main phases of work on classification with the statistical environment. The first, the classical phase in which researchers have concentrated of Fisher's early work on linear discrimination. The second, the modern phase, in which more flexible classification models have been developed. These models have provided an estimate of the joint distribution of the features with each class in order to develop an efficient classification rules.

C. Feng and D. Michie have introduced the second approach, namely machine learning approach. He has focused totally on decision-tree approaches, in which classification is derived from a sequence of logical steps. He has addressed the most complex classification problems and proved that the decision trees could solve theses kinds of complex problems in the case of existing sufficient data. The major disadvantage specified is the large amount data that should be available for implementing this approach.

R. Rohwer, M. Wynne-Jones and F. Wysotzki have addressed the third approach, namely neural network approach. He introduced many techniques in implementing neural network approaches. It was also identified how the neural network approaches combine the complexity of some of the statistical approaches with the machine learning objective of imitating human intelligence. He has also shown how this process is hidden in the hidden layers and how it is so difficult to let the learned weights transparent to the user. He has also discussed different techniques to improve all types of feed-forward networks. This is focused on changing the network topology as training proceeds.

Jiawei Han and Micheline Kamber (2001) introduced in their book "Data Mining: Concepts and Techniques" some classification techniques that are suitable for the data mining technology. They have combined the classification process and prediction process together to be considered as a unified process in many of today's businesses. They have identified that the data classification process consists actually of two related phases, namely the learning phase and the testing phase. It was noticeable that they have focused on how to increase the efficiency of the classification process through preparing the data for classification in a process called data preprocessing, which in turn contains many methods such as data cleaning, relevance analysis and data transformation. They worked with different classification algorithms, but have concentrated on the decision tree and backpropagation algorithms. At the end they introduced some valuable techniques to measure the accuracy of the produced classifiers.

Isabella Guyon and André Elisseeff (2003) have contributed in the data classification field with a very valuable study on variable and feature selection. They have provided a very efficient description for feature construction, for feature ranking and for feature validity assessment methods. They even have summarized the steps needed to solve a feature selection problem. Variable ranking process has taken a very important part in their study and they have introduced many different methods that would help in the process of variable ranking. In addition, different methods that facilitate the process of feature construction and the process of reducing the space dimensionality. In many applications, reducing the dimensionality of the data by selecting a subset of the original features may have many advantages. This is so important when minimizing the expenses of making, storing and processing measurements is of concern. If these factors are not that important, other methods of reducing the dimensionality should be tackled.

Janez Brank, Marko Grobelnik, Nataša Milić-Frayling, Dunja Mladenić (2002) have proposed a new technique for feature selection based on Support Vector Machines (SVM) which are a set of related supervised learning algorithms, applicable to both classification and regression. Their experiments showed that the SVM-based feature selection has preserved the classification performance while dramatically reducing the size of the feature space and increasing the scattered data. Their strategy was first to train linear Support Vector Machines on a subset of training data to build initial classifiers, in the next step they have eliminated the features which have low weights in order to specify a certain level of data sparsity. They have defined sparsity in their work to be the average number of non-zero components in the vector representation of data. The second step represents the process of feature selection in their study. In the last step, they have created a representation of the full training dataset by using the retained features only. Then they have retrained the linear SVM classifier in the derived feature space and used the final results in the testing process. The method used in their work was designed totally to take advantage of the free memory that has been derived by the increased data sparsity. It was also designed to include large training datasets while keeping the memory consumption at a constant rate.

Sheng MA and Chuanyi JI (1999) have reviewd recent enhancements in supervised learning with a deep focus on performance and efficiency of learning algorithms. They major concentration was on using a special type of adaptive learning systems that are based on neural network architecture. Their valuable study they introduce four learning approaches, namely training on individual model, combinations of several models that have been trained efficiently, combinations of several weak models and evolutionary computation of models. In addition they have showed the advantages and disadvantages of these approaches. Through their study, it has been clear that in order to have an efficient adaptive learning machines, a good performance is required. The most important result of their study was about using combinations of weak classifiers, which use an incremental combination scheme and a randomized algorithm. This approach has shown the potential to achieve time efficiency and an efficient generalization performance.

Sholom M. Weiss and Casimir A. Kulikowski have introduced a practical guide to classification learning algorithms and their applications. It was shown clearly how these computer systems learn from sample data and then could make prediction for new cases. Practical learning systems from different fields such as statistical pattern recognition, neural networks and machine learning have been presented. In addition, the hidden concepts of learning process, its strengths and weaknesses and their future performance have been widely discussed. Moreover, they have offered some valuable recommendations for selecting learning algorithms such as nearest neighbor, backpropagation and decision trees. In general their book gives a consistent introduction to many learning algorithms and covers techniques that estimate a classification model's accuracy.

The nearest neighbor classification rules, which relates a new unclassified sample to the nearest previously classified samples, is a nonparametric statistical technique, in which the classification processes is independent of the underlying joint distribution on the given data samples (T. M. Cover and P. E. Hart (1966)). They have introduced in their study a complete analysis of the nearest neighbor technique based on a statistical point of view. A comparison between the probability of error of the nearest neighbor rule and the bayes probability of error, which represents the minimum probability of error over all decision rules taking underlying probability structure into account, showed that the probability of error of the nearest neighbor simple rule is less than twice the bayes probability of error. Even it is less than the twice the probability of error of any other decision rule, whether it is nonparametric or otherwise. This fact has been proved when using large dataset.

C. Domeniconi, J. Peng and D. Gunopulos (2002) have proposed an adaptive nearest neighbor classification technique that helps greatly in minimizing estimation bias in high dimensions classification problems. The curse of dimensionality was the fundamental motive that made them work hard in this study. As a result, they have estimated a flexible metric for computing neighborhoods based on Chi-squared distance analysis. This metric depends totally on query locations in the feature space. In addition, the resulting neighborhoods are spread along less relevant feature dimensions and tightened along most powerful ones. The result was having the class conditional probabilities to be more homogenous and smoother in the modified neighborhoods, so that improved classification performance can be reached.

Euihong (sam) Han, George Karypis, and Vipin Kumar (1999) have focused on text classification problems in their work. As the task of classifying different documents in a certain field into predefined categories or classes is challenging due to the large amount of documents exist and the curse of high dimensionality nature of documents datasets, they have started their efforts in order to develop a new algorithm that could help in classifying documents in a very efficient way. In addition, the problem in the current widely used algorithms such as C4.5 and RIPPER, which do not work well with large number of features, have led them towards their valuable study. They have proposed a new algorithm that is based on the nearest neighbor model. The new developed algorithm is called weight adjusted *k*-nearest neighbor (WAKNN). The core of this algorithm is learning the importance of each word in the training document set and the weight vector reflecting this importance is then maintained. The nearest neighbors for a certain document are then computed based on the matching words and their weights. They have implemented many experiments on several synthetic and real life datasets, which showed the high performance of the algorithm compared to other algorithms. Their experiments with synthetic datasets proved that this algorithm is robust under certain emulated conditions, but the empirical results on real word documents demonstrated that this performance of this algorithm is better than the performance of some other classification algorithms such as C4.5, RIPPER and Rainbow.

Sameer Singh, John Haddon, Markos Markou (1999) have presented a new model that has developed the basic definition of the standard nearest neighbor algorithm to include the ability to resolve conflicts when the highest number of nearest neighbors are found for more than class label. In addition, they have proposed a new nearest neighbor model that is based on finding the nearest average distance rather than nearest maximum number of neighbors. These two new models and their performance have been explored and evaluated based on image understanding data. In order to implement the first model, they have used two stages. In the first stage they have declared a certain given class to be a winner, if and only if the given class has more training data samples closer to the testing sample. In this case the testing sample is classified to belong to the given class. The second stage, which resolves the conflict, takes place, if for more than two classes surrounding the testing sample, an equal number of highest neighbors has been found. In this case, the winner class, whose distance from testing data averaged over all its training data samples, is the smallest. In the second model, they have not considered the number of neighbors but only the average distance of classes from testing data. The winner class here is the one with the smallest distance from testing data. In this case the testing sample is assigned to this class. When noise testing data have been used in their study, the new nearest neighbor models showed very efficient and promising results for further studies when compared with neural networks.

Many researchers have taken place in order to develop intelligent systems based on different techniques and especially Von Neumann's architecture, but these tries haven't achieved the desired success needed. Afterwards, different studies inspired by the biological neural networks have been made, so that various scientific disciplines have implemented designs for artificial neural networks to solve a variety of problems in decision making, prediction and classification (Anil K. Jain, Jianchang Mao and K. Mohiuddin (1996)). They have introduced their work to be considered as a tutorial of artificial neural networks. They have also discussed the motivations that were the reason behind developing artificial neural networks. They have introduced great information about how the artificial neural networks could be implemented in supervised and unsupervised learning. In addition, different learning algorithms and rules have been tackled widely. In their work, the basic neural networks models for supervised learning, such as single-layer perceptron, multi-layer perceptron have been presented. On the other hand, they have discussed different models for the unsupervised learning, such as self organizing maps. A valuable section for the applications that could be developed using different artificial neural networks has been introduced.

John Moody (1994) has introduced a valuable study on artificial neural networks. He has totally focused on describing risk estimation and network topology selection. In his work, we will find that he has defined risk estimation as the expected performance of an estimator in predicting new observations. Risk estimation process could be used in estimating the quality of the predictions derived by the selected neural networks model and in model selection. He has also proved that these two processes are so important in classification problems that have limited data samples. His study has introduced two approaches for estimating prediction risk, namely data resampling algorithms such as general cross-validation and nonlinear cross-validation and algebraic formulae such as the predicted squared error and the generalized prediction error. As a result for his work, intensive search over the space of network architecture was showed to be computationally infeasible even for networks with moderate size. In the process of network architecture selection, he has concentrated on the methods of selecting the number of hidden nodes in hidden layers in the multilayer perceptron model. The algorithm used here is the sequential network construction which builds a sequence of networks. Each one of these networks is fully connected and uses all input samples. The only difference is the number of hidden nodes, so that not all of the networks have the same number of hidden nodes.

Warren S. Sarle (1994) has proposed artificial neural networks from a different perspective, namely the statistical approach of implementing neural networks. He has also described how artificial neural networks could process vast amounts of data in such accurate way as many other statistical techniques. In addition, the learning process in artificial neural networks is the same as in many statistical algorithms. So, if ANNs are intelligent, then many statistical methods must also be considered intelligent. In his work, Sarle showed the relationship between neural networks and statistical models such as generalized linear models, polynomial regression, nonparametric regression and cluster analysis. He also identified how the neural networks learning algorithms are inefficient because they have been designed to be processed on massively parallel computers but have been actually implemented on common serial computers such as PCs. Another reason he has introduced as an evidence for the weak efficiency of neural networks algorithms is that they have been designed for systems whose data are not stored or whose data are transient. In statistical methods, the data are usually stored, so they are more efficient the neural networks algorithms. He has related the two famous neural networks models, namely the single-layer perceptron and multi-layer perceptron, to their equivalent statistical model. Sarle has also described how the neural networks and statistics are not competing methodologies in the data analysis field. Both methodologies are overlapped to some extent. Statistical methods are directly applicable to neural networks in a variety of ways, including estimation criteria, optimization algorithm and confidence intervals. As a final result, he found that a shared communication between neural networks and statistics would benefit both.

Joseph Reisinger, Kenneth O. Stanley and Risto Miikkulainen (2004) have worked on a very promising neural networks approach that automatically evolves network topology and weights. This approach has been shown to be powerful in nonlinear optimization classification problems. Because the curse of dimensionality has been currently tackled and has become widely spread, they have introduced this approach to address this problem. The dimensionality curse is identified in their work by two factors: the number of inputs and outputs a network has and the architecture complexity of that network such as the amount of hidden nodes in the hidden layers. They have also proved that the object space may be too large to be searched efficiently, if the input, output or network topologies are high dimensional. They have identified a valuable method for tackling the dimensionality curse by breaking the classification problem down into simpler sub-problems to be solved in parallel. In this case the evolution process could be implemented more efficiently. A new method, called NeuroEvolution of Augmenting Topologies (Modular NEAT), has been developed to automatically perform this decomposition during the evolution process.

Marvin L. Minsky and Seymour A. Papert (1969) published their work in a very decent book that analyzes the perceptron theory in details. Before their work the neural networks field was in frustration phase, as the basic perceptron theory was a new concept. Minsky and Papert summed up this general feeling by the representational limitations of perceptron theory. Their arguments were very influential in the field and accepted by most without further analysis. They introduced the XOR problem which delayed any further studies in the neural network field for about ten years. Minsky and Papert's book was the first example of a mathematical analysis carried far enough to show the exact limitations of a class of computing machines that could seriously be considered as models of the brain. They focused on the problems of learning. They showed that as various predicates scale, the sizes of coefficients can grow exponentially, thus leading to systems which need unlimited cycles of a convergence process.

Steven L. Salzberg (1997) introduced a comparative study for some classification techniques. He concentrated on the quality of experimental designs and how it could result in statistically invalid conclusion, if the required experiments have not been designed very carefully. This problem is clear when using very large datasets. In his work, Salzberg has described several incidents that can invalidate an experimental comparison, if they are ignored. In addition, he has also proved that these incidents and their conclusions apply to classification and also to computational experiments in the classification fields. He has also described how comparative studies are very important when evaluating some types of algorithms is needed and presented some solutions about how to avoid drawbacks of implementing different experimental studies. He tackled a very serious problem, namely how could one select which algorithm to use for solving a new classification problem? In his study, Salzberg has addressed the methodology that could be used to answer this question and discussed how this problem was tackled in the classification field. The target of his work was not to discourage comparative analysis, but the target was to help researchers focus clearly in designing a comparative study.

W. Schiffmann, M. Joost and R. Werner (1992) have identified how most artificial neural networks are not structured. Their architecture depends totally on the human designer's insight or vision. This designer decides how many neurons should be used in each layer of the layers that construct the network topology. Actually, they have added a new definition for the neural network architecture, namely directed graph. The neurons could be considered as the nodes and the connections which have the weights could be then considered as the edges of this directed graph. So, finding problem-adapted architecture is equivalent to optimizing the topology of the underlying graph. In their work, they have presented many approaches for automatic topology optimization of MLP with backpropagation neural networks that could be adapted automatically depending on the classification task complexity. In addition, they have presented the benefits of working with adapted network architecture for software simulations and for the hardware implementation of artificial neural networks. The most important benefit is the enhancement in the learning process as it becomes accelerated because fewer connections must be trained per epoch. In general, they have considered the process of optimizing network architecture is as well important as developing a new learning algorithm.

Kenji Fukumizu (2000) has tackled the problem of active learning in multilayer perceptrons as the neural networks performance would be significantly improved if the training data are selected actively. He has introduced a way to prepare a probability distribution using the proposed method of active learning. As a result, he has then obtained training data samples from the distribution specified. The core of this methodology was to develop an information-matrix-based criterion. The most critical problem he faced is that the required inverse of an information matrix may not exist. His proposed method is applicable to three-layer perceptrons. The proposed active learning technique has a method for reducing the hidden neurons in the hidden layers. The reduction procedure used eliminates redundant hidden nodes during the learning process and keeps the information matrix nonsingular. In his method, he considered the reduction criterion is so important, because extreme elimination of hidden nodes may degrade the approximation capacity and increase the mean squared error.

Alan F. Murray and Peter J. Edwards (1993) have analyzed the effects of analog noise on the synaptic weights during the learning process in the multilayer perceptron networks. This analysis showed by mathematical expansion and by simulation that injecting random noise to network weights during learning process improves fault tolerance without additional supervision. They have also showed that the states of hidden nodes and the learning path have been adjusted in a way that enhances the learning quality and performance and the time required for completing the training process. They have used the intermediary influence of noise to distribute the information optimally across the weights. Their implemented technique has generated more robust internal representations which in turn have resulted in better generalization to the small differences in the characteristics of the testing data samples. In previous studies, it was known that any inaccuracy during training process is unfavorable and harmful to the learning process in multilayer perceptrons networks. They have proved in the study that the analog or continuously changing inaccuracy is not that harmful. The final results of their experiments proved to be efficiently general for all training datasets where weights are updated incrementally.

Martin Riedmiller (1994) has introduced the concept of supervised learning in multilayer perceptrons based on the technique of gradient descent, on which the backpropagation algorithm was built. He discussed some of the important pitfalls and problems of the backpropagation learning algorithm. In addition, Riedmiller has focused on adaptive learning methodologies with a clear description of some of the most popular learning algorithms according to their classification in terms of global and local adaptation strategies. For the global adaptive techniques, he introduced some algorithms such as Steepest Descent and Conjugate Gradient Descent. On the other hand, he introduced some algorithm such as the Delta-Bar-Delta Rule, SuperSAB, Rprop and Quickprop. In general, he introduced an overview over past and recent milestones in developing supervised learning algorithms for the classification techniques based on MLP. Many of the proposed techniques, especially the global adaptive techniques, are based on schemes taken from many dimensional optimization theories which in turn need complex computations. Using some experiments, he proved that the local adaptive algorithms, such as Quickprop and Rprop, have a significantly faster convergence rate than the ordinary gradient descent algorithm.

Devin McAuley (1997) and Simon Dennis (1999) introduced a very valuable tutorial about the backpropagation algorithm. They have tackled this algorithm because they have considered it to be the most appropriate method for classifying predicting reasonable information from scattered, noisy or incomplete data. They have described the topology for the neural networks that implement the backpropagation algorithm and in addition they have discussed the architecture of the algorithm itself with a big focus on step that should be followed to run the algorithm. They have then identified the learning process in the backpropagation algorithm in two steps. First each data sample is presented fed into the network and propagated forward to the output layer. Second, a method called gradient descent is used to minimize the total error in the training dataset. A very important and valuable part in their study is the methods introduced to help in selecting the initial weights and in when to update the weights during the learning process.

Mohd Yusoff Mashor (2000) has introduced an important comparison between backpropgation, recursive prediction error and modified recursive prediction error algorithms for training multilayer perceptrons networks. In his study, he has investigated the performance of these algorithms and implemented real life experiments using real life data to prove his findings. Training MLP networks with backpropagation algorithm could provide satisfactory results. However this technique is considered to be the basis to the neural networks researches. He identified the backpropagation algorithm to belong to the steepest descent type algorithm. This fact made the backpropagation algorithm to suffer from a slow convergence rate. On the other side, the recursive prediction error and its modified version are a Gauss-Newton type algorithm that has a better convergence rate than the steepest descent type algorithm. His study proved that the neural network trained using the backpropagation algorithm did not have good generalization. This is because the prediction process in the testing dataset is not as good as in the training dataset.

Scott E. Fahlman (1988) has described the results derived during the first six months of his study. He has introduced a systematic and empirical study of learning speed in backpropgation networks. He measured the algorithms used against some of benchmark problems in order to develop faster learning algorithms with high performance rate and to contribute in developing a methodology that will be of a great benefit to future studies in this field. The study resulted in having a new learning algorithm, which is considis faster than standard backpropagation by an order of magnitude or mered to be a combination of several ideas. The new algorithm ore and appears to scale up much better than standard backpropagation algorithm as the size and complexity of the learning task grows. This result is encouraging, but is not conclusive because Fahlman has only tested the new techniques against a very small set of benchmark problems.

Survey of Available Classification Techniques

As the data analysis field is evolving to a large extent, many classification techniques have been developed to help in improving efficiency and scalability. In this section we try to introduce a strategic survey on different classification techniques in order to achieve the objectives of this study. In this survey we will be focusing on the following criteria which will help us evaluating each technique:

- Predictive Accuracy: This refers to the ability of technique to correctly predict the class of a new data sample.
- Speed: This refers to the computation costs need when implementing the technique
- Scalability: This refers to the ability of the technique to classify the samples correctly and efficiently when using large and complex datasets.
- Robustness: This refers to the ability of the technique to correctly predict new samples when using noisy data.

Classification by Decision Tree

Decision tree is a form of flow diagram in which a series of selection criteria classify the data into subcategories. In order to classify an unknown sample, the feature values are tested against the decision tree, and then a path is traced starting from the root to a certain leaf node that predicts the class label for that sample. Decision trees can be easily converted to classification rules.

ID3 algorithm and its extension C4.5 algorithm are well-known decision tree induction algorithms which are considered to be greedy algorithms. Using these algorithms, the tree starts as a single node which represents the training samples. This node becomes a leaf, if all the samples belong to the same class. Otherwise, the algorithm uses the information gain measure to select the test attribute at each node. This test attribute will be used to split the tree into individual classes. Then, a branch is created for each known value of the test attribute and the data samples are organized accordingly. The algorithm implements the same process repeatedly to build a decision tree. In ID3 all the attributes should be categorical or discrete-valued and continuous-valued attribute should be discretized. In C4.5 this has been solved as one can use continues-valued attributes without discretization.

In the process of building a decision tree, many of the branches will reflect irregularities in the training data due to noise or outliers. This fact decreases the efficiency of the decision tree. Tree pruning techniques can help in solving this problem which could be called as overfitting in the data. Such techniques are considered to be statistical methods that remove the least reliable branches in order to improve the classification performance and to make the classification process faster.

During the process of repeatedly splitting the data into smaller and smaller categories, classification by decision tree induction faces three important problems, namely fragmentation, repetition and replication, which can minimize the scalability and the efficiency of the classification process. In fragmentation the number of data samples at a given branch becomes so small that these data samples are statistically insignificant. Repetition occurs when an attribute is repeatedly tested along a given branch of the tree. In replication, duplicate subtrees exist within the tree. All these problems can decrease the scalability and the performance of the resulting tree. Many methods have been introduced to solve these problems, but they require more computations and more time to be implemented. So it may slow the speed of training.

It has been proved that the currently existing decision tree algorithms, such as ID3 and C.45, are only efficient and scalable when using small datasets. But when using very large real-world datasets these algorithms might face deficiency in scalability. Most of these algorithms must store all of the training samples before the classification process starts. This fact limits the scalability of such algorithms, as the building the decision tree could become inefficient because of the swapping of the training samples in and out of main and cache memories. Many recent algorithms, such as SLIQ and Sprint, have been proposed to address the scalability issue. Although these algorithms handle disk-resident datasets that are too large to fit in the memory, SLIQ uses its memory-resident data structure, so that its scalability becomes limited. SPRINT has the power to remove all memory restrictions and then requires the use of a hash tree matching the size of the training dataset. But if the size of the training dataset grows, the process will experience expensive computation costs that will decrease the performance of the classification.

Bayesian Classification

Bayesian classifiers are statistical classifiers. They can predict the probability that a given data sample belongs to a certain class. This type of classification is based on the Bayes theorem. In Bayesian classification, two types have been introduced, namely naive bayesian classification and bayesian belief networks.

A naive bayesian classifier, known as Idiot's Bayes, is a very simple probabilistic classifier. It is totally based on probability models that contain strong independence assumption. In general, it assumes that the impact of a feature value on a given class is independent of the values of the other features. This assumption called class conditional independence. It has been developed to simplify the computations involved in the classification process.

The bayesian belief networks specify joint conditional probability distributions for all features in the given dataset. Actually, they define class conditional independencies between subsets of the features. In addition, they provide a graphical model of casual relationships, which can be used in the learning process. Many methods can be used to train a Bayesian belief network: the network architecture can be given in advance or they can be derived from the given data samples. The network attributes or features could be observable or hidden in all or some of the training data samples. Hidden data can also be considered as missing data. In the case of knowing the architecture and of having observable attributes, the network is trained simply and straightforward. When the architecture of the network is given in advance and some of the attributes are hidden, then a method of gradient descent may be used to process the learning, it is like training an artificial neural network. So the objective here is to find the most accurate weights that will help in classifying the data correctly.

Theoretically, Bayesian classifiers have the least error rate when compared to all other classification techniques. However practically, this is not always the fact because of the inaccuracies in the assumptions derived by them. Many studies have found that this type of classification is comparable to decision tree and neural network classifiers in some domains. Although bayesian classification introduced high accuracy and speed when implemented using large datasets, it is not an accurate technique in some critical classification tasks, such as medical analysis as this field can't be studied based on assumptions like the ones made by Bayesian classifiers. In addition, bayesian classifiers are more difficult to tune because they need more storage and expensive computation costs. They are considered to be parametric classifiers, so that they are valid only in normal data distributions as the nearest neighbor technique.

Case-Based Reasoning

It is known that the nearest neighbor technique stores all the training samples before building a classifier. But case-based reasoning technique stores training samples which are complex symbolic descriptions, so it is considered to be a lazy learner that needs to store all the cases before building a classifier. CBR has been applied to many areas, such as engineering, where cases are technical designs, and law, where cases are legal rules.

A case-based reasoner checks first if an identical training case exists, when a new case needs to be classified. If one exists, the related solution to that case is returned. But if no case exists, the CBR searches for training cases similar to the new case. These training cases could be considered as neighbors of the new case. Then the CBR tries to combine the solutions of these neighbors to formulate a final resolution for the new case. So, it is clear that CBR needs to have background knowledge and strategies for solving problems in order to introduce an efficient solution for the new case. It is more like an expert abstract technique that need intensive studies to improve its process.

Decision tree induction and case-based reasoning are complementary approaches, as induction compiles training data samples into general knowledge, where CBR directly interprets those training samples. So, both techniques compliment each other. A combination of both techniques may be used to achieve an efficient and accurate classification process.

Genetic Algorithms

Genetic algorithms are built by a process of natural evolution, which imitates the development of biological systems. The genetic algorithms work as follows:

- Create an initial population of solutions or rules coded as artificial chromosomes.
- Select the best solutions for recombination of the mating chromosomes.
- Perform mutation and other variation operators on the chromosomes.
- Use these offspring to replace poorer solutions

It has been proved theoretically and empirically that genetic algorithms lead to improved solutions in different domains. The most important part here is designing a genetic model that helps in solving complex classification tasks. Genetic algorithms can be combined with neural network techniques to solve more complicated problems. They can be used to evaluate the efficiency of other classification techniques.

Rough Set Approach

Rough set theory depends on the formation of equivalence classes in the given training dataset. All the data samples establishing an equivalence class are identical with respect to their attributes. Rough sets can be used to roughly specify classes that can't be distinguished in terms of the available attributes. A rough set definition for a given class *P* is approximated and specified by two sets, namely the lower approximation of *P* and the upper approximation of *P*. The lower set consists of all of the data samples that certainly belong to *P* without ambiguity. The upper set consists of data samples that can't be identified as not belonging to *P*.

Rough set theory can be used in classification processes to help in discovering structural equivalences with some given noisy data. It works properly when using discrete-valued attributes. Continuous-valued attributes must be first discretized before implementing. It also can be used in feature reduction, where the attributes that have the least significant contribution towards the classification of the given training samples can be identified and removed. This theory can also be used for relevance analysis, where contribution of each attribute is evaluated and measured with respect to the classification process.

Rough sets have an advantage over bayesian classifiers: no assumption about the independence of the attributes is important nor is any background knowledge about the data. In real world classification tasks, the given datasets are usually uncertain and complete. So, two approximations as proposed by rough set theory won't be sufficient to identify the information hidden in the datasets. As a result, the theory has been adjusted to deal with the complex datasets. This change has made rough sets to lose their original features and to be more like fuzzy sets.

Fuzzy Logic

Rule-based approaches for classification are so strict and sharp in applying the rules. For example, if we have a rule that says: customers who have had a job for two or more years and whose salaries are $50K or more can apply for a credit card. So, if there is a customer who has had a job for more than two years, but his/her salary is $49.5K, then this customer won't be able to get a credit card. This is totally unfair. Actually, fuzzy logic can then be implemented in the rule-based system to allow fuzzy thresholds to be defined.

Rule-based systems introduce a sharp cutoff between classes, where fuzzy logic used truth values in the range between 0.0 and 1.0 to represent the degree of membership of an attribute value in a certain class. This approach helps greatly in classification processes implemented by rule-based systems as it provides a high level of abstraction. Fuzzy logic implementations require complex calculations and may experience high computation costs.

The Techniques implemented in this Thesis

We have selected to work on three techniques, namely nearest neighbor technique, single-layer perceptron technique and multi-layer perceptron technique. In classification field no one can guarantee that a certain technique is the best one. This may be proved by implementing real world datasets to test efficiency and scalability of a certain technique.

Nearest neighbor technique is considered to be the most simple and straightforward technique that generates very high accuracy, as it will find at least one data sample near to the unknown. It has been used widely in classification, clustering, feature extraction, although it requires storing the entire training samples. This disadvantage may lead to slower recognition speed but more accurate. In some classification task, nearest neighbor is the most efficient technique to use. For example, if we have simple task that works on a small dataset, there is no need to implement advanced techniques, instead nearest neighbor technique will suffice.

The other two techniques, namely single-layer perceptron and multi-layer perceptron, are artificial neural networks techniques. Neural network are tolerant to noisy data and they have the power to classify patterns on which they have not been trained. So theoretically, neural networks are the best classification technique, but practically no one can guarantee the efficiency before empirically implementing the technique. The single-layer perceptron and the multi-layer perceptron technique do not require storing the training samples; instead they learn the weights of the network architecture and then use them to classify the testing samples. This approach speeds the recognition time. The multi-layer perceptron is a very efficient and scalable technique that can classify complex data samples which are not linearly separable. In general, with some more studies, neural networks can be one of the best techniques in data classification.

CHAPTER III

RESEARCH DESIGN AND METHODOLOGY

The methodology applied in this thesis is based on analyzing some techniques for data classification focusing on performance issues and conclusions derived from various experiments which implement each technique.

Rationale for the Thesis

A systematic approach is implemented in order to present each classification technique in this thesis: first, the architecture and concept behind each classification technique is investigated widely. Second, the algorithm for each technique is presented in a logically organized way. Third, a real life classification problem is examined in order to reach the best solution that will be used to classify the given data samples and the new data samples properly and efficiently. Fourth, a final discussion about performance of each technique is presented according to the results obtained from the experiments. The rationale for this study is to find the efficient ways that would help in selecting the best classification technique in relation to the introduced task.

Datasets and Data Collection

In this thesis, three datasets have been used to implement the experiments. The datasets have been previously prepared to reflect the nature of each classification technique used in this thesis. The first dataset, which has been used for the nearest neighbor technique, represents four sports players' classes: judo players, jockeys, basket ball players and rugby player. This dataset contains three columns: the first column represents the weights of the sports players, the second column represents the heights of the players and the last column represents the class labels which will be the differentiator between the four classes. Each record in the dataset represents one sport player who belongs to one of the four classes. If the player is identified by class label 1, then this player belongs to the jockeys' class. If the player is identified by class label 2, then this player belongs to the basket ball players' class. If the player is identified by class label 3, then this player belongs to the rugby players' class. If the player is identified by class label 4, then this player belongs to the judo players' class. This dataset contains 112 records.

The second dataset, which has been used for the perceptron technique, represents two sports players' classes: jockeys and rugby players. This dataset is the same as the first dataset but with only two classes. So the first column is the weight column and the second column is the height column. The third column is the class label. The jockeys are identified by class label -1 and the rugby players are identified by class label +1. This dataset contains 56 records.

The third dataset, which has been used for the multilayer perceptrons technique, is actually divided into two separate datasets, namely the training dataset and the testing dataset. These dataset represent five classes of cars, namely small cars, midsize cars, compact cars, large cars and sport cars. Each one of these dataset has 250 records and 15 columns. The first 14 columns represent some information that helps in differentiating the cars from each other. These 14 columns are: Price, City MPG, High Way MPG, # Cylinders, Engine Size, Horse Power, RPM at max HP, Revs/min, Fuel Tank Capacity, Passenger Capacity, Length, Width, Wheel Base, and Weight. The 15th column is the class label. The class of small cars is identified by class label 1, the class of midsize cars is identified by class label 2, the class of compact cars is identified by the class label 3, the class of large cars is identified by class label 4 and the class of sport cars is identified by class label 5.

After finishing the analysis of each algorithm, we start to test the efficiency and scalability of the multi-layer perceptrons technique by using a very large and complex dataset that consists of 617 features and 26 classes. This dataset contains information about isolated spoken letters that need to be classified into 26 classes (one per letter).

Implementing Tool

MatLab has been used intensively to implement the three algorithms used in the experiments. As the most important techniques in this thesis are the single-layer perceptron and the multi-layer perceptron, MatLab environment has dedicated components which facilitate the process of implementing artificial neural networks without the need for developing a black box which constructs the network. Based on only few lines of code MatLab has the power to train any artificial neural network with any topology required. Once the network is trained the optimum weights have been learnt, we can then execute a certain code on the testing dataset and obtain the final accuracy for the testing process. For the first technique, namely the nearest neighbor technique, we have developed a program based on MatLab's syntax that facilitates the implementing of the nearest neighbor method. Once we run the code against the new unknown data sample, we obtain the class label, to which this unknown sample belongs. In general the MatLab environment is very efficient when working on such experiments, because of the easyhandled keywords that contain very concentrated logic implemented properly to solve very complex classification problem.

Classification algorithms as Input/Output Mapping

In order to be able to classify data based on different techniques we have to introduce and define some concepts that have been fully integrated in the process of supervised learning (see figure 1).

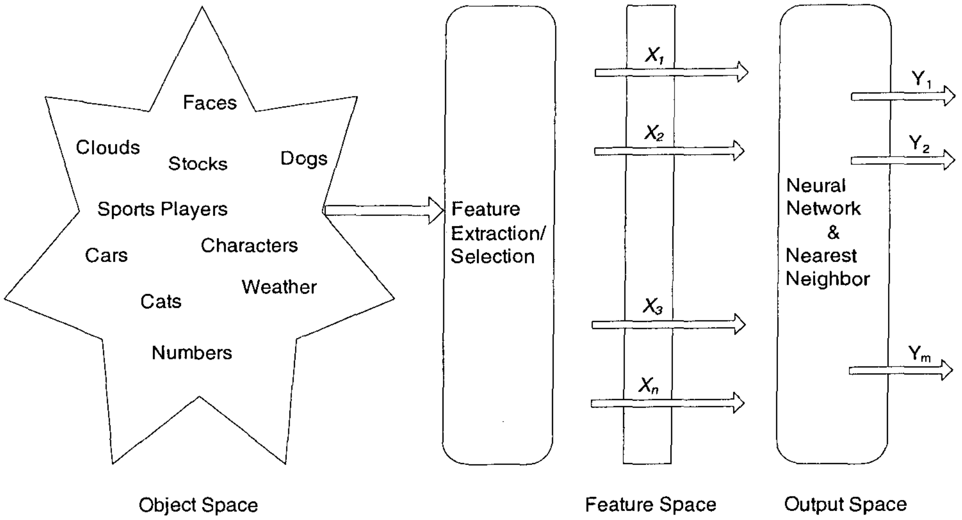


Figure 1. Input/Output Mapping

Object Space represents the general field that needs to be processed to classify its given members, for instance; in the cars object space each single car represents an object with some identifying attributes (features) that distinguish a certain car from others, such as: model, manufacturer, year of production, length, width, height, color, wheelbase, kerb weight, fuel system, displacement, horse power and many others. So, the object space of cars contains many different and similar cars. Before anyone can classify a data, he/she should first specify the object space to be able to proceed to the feature selection process.

Without specifying an object space there will no be a base to launch the classification process.

Feature is any distinctive aspect, quality or characteristics that distinguish objects in the same object space. A feature may be symbolic (i.e. color) or numeric (i.e. length)

Feature Extraction/Selection is the process of identifying the most qualitative attributes of the objects that helps to solve the targeted classification problem. It has been known that extracting the relevant attributes for decision making is considered as one of most important problems that may face data classification. In addition it has been proved that minimizing the dimensionality of the pattern representation through reducing features and removing redundant and irrelevant measurement improves the performance of supervised learning and as a result the data classification process will be significantly faster [11].

In the cars object space, if we are trying to classify some cars to be in five classes such as: small, med size, compact, large and sporty, the color and the production year will hardly help in resolving the problem. But other attributes such as: length, height, weight, horse power and wheelbase may be of a great impact to solve the given classification problem. Feature selection needs solid experience, many different mathematical functions and different probability theories to identify the most qualitative feature space. In many classification problems a trial and error technique may lead to the best features selected. The process of feature selection reflects many potential benefits that will help in solving the classification problem in a more reliable way. Some of these benefits could be: facilitating data visualization, reducing the measurements and storage requirements, minimizing training and utilization time. So, the process of feature selection plays a very important role to solve any classification problem. The more relevant the selected features are, the faster and the more efficient the given classification problem is.

Feature/Input Vector is the combination of *n* features represented as an n- dimensional column vector. After the process of feature extraction the produced and identified features represent a feature vector that will act as the input data for the classification algorithms (the black box). The feature vector defines the n-dimensional space known as feature space. In simple classification problems that have only two-dimension feature vector the objects could be represented in the feature space. This representation is known as a scatter plot.

Output Vector is the combination of *n* desired output values (class labels) represented as an n-dimensional column vector. The output vector defines the n-dimensional space known as output space. The output vector is the target of any classification problem. The classification algorithm processes the given inputs internally and produces the actual output that might be far of the desired output. In this case new process is started until reaching the desired output.

CHAPTER IV

DATA ANALYSIS AND FINDINGS

Nearest Neighbor Classifiers

As we introduce some data classification techniques using different algorithms in this thesis, we prefer to start the technical cycle with the simplest algorithm, that is, Nearest Neighbor classifier. This algorithm is used widely in most simple classification problems where it is required to classify unknown samples based on historical known samples (see figure 2).

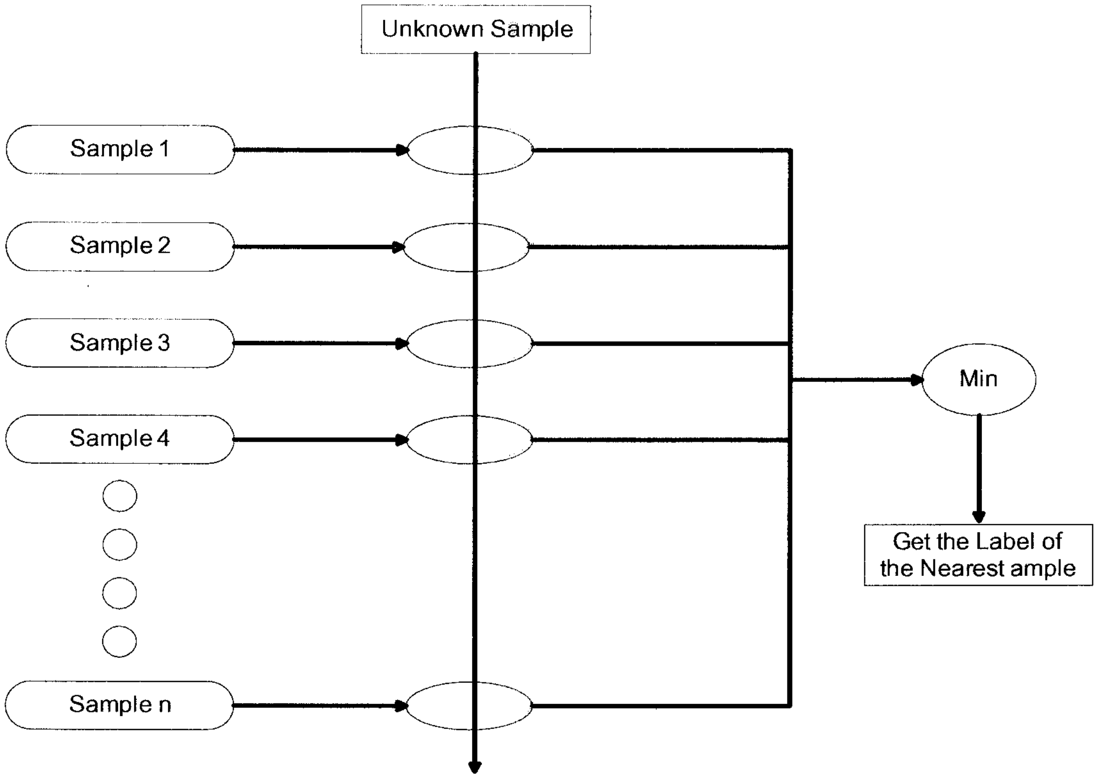


Figure 2. Nearest Neighbor Classifier

Architecture of the Nearest Neighbor Classifier

The Nearest Neighbor algorithm is based totally on supervised learning by similarity. The core motivation behind the Nearest Neighbor classification technique was based on the idea: “things that look alike must be alike”. The Nearest Neighbor known as k-NN is a technique that classifies each unknown sample in a given dataset based on the class labels of the historical known k sample(s) most analogical to it. In any classification problem we may face there are two boundaries that must be possessed during the classification process. These two boundaries are: The person who is in charge has a complete knowledge about the underlying joint distribution between the observations (data samples) and the desired output categories (class labels), or he/she may have no knowledge about this underlying distribution except what can be assumed from the samples themselves. The Nearest Neighbor classifier is based totally on the second boundary as it is a nonparametric classifier.

As we cannot ensure the type of the distribution, it is acceptable to assume that the data samples which are close to each other belong to one class or category. So, in order to determine which data samples are grouped together in one category, we can use the simplest way of measuring the distance between the unknown sample and it's near historical data samples known as k.

Because the Nearest Neighbor classifier depends on computing the distance between the unknown samples and the historical samples, we have to assure that all the samples are numeric and not categorical or symbolic. The historical data samples are identified by n-dimensional numeric features. So, each historical sample is stored in an n-

dimensional feature space in the form of {x,Ω}; where x represents the feature vector and Ω represents the pattern class.

When a new unknown sample is needed to be classified towards the existing class labels, the k-NN searches the feature space for the k historical data samples that are closest to the unknown sample. Each historical sample and each unknown sample is represented as feature vector where the feature vector of the unknown sample is $\underline{X}$ and the feature vector of the historical sample is $\underline{Y}$ and the number of features is n as shown here below

$$\underline{X} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \qquad \underline{Y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \cdot \\ \cdot \\ y_n \end{bmatrix}$$

The distance between the two vectors could be measured using the Euclidean distance as shown in the following equation:

$$D(\underline{X},\underline{Y}) = (\sum_{i=1}^{n} | x_i - y_i |^2)^{1/2}$$

The distance between the two vectors could also be measured using the Manhattan distance (City Block) as shown in the following equation:

$$D(\underline{X},\underline{Y}) = (\sum_{i=1}^{n} | x_i - y_i |)$$

At this point we can say that the Nearest Neighbor classifier is a very simple and straight forward classifier that can be widely used in different classification problems.

Practical Experiment: MATLAB Code

In our given classification problem the target is to classify a new unknown sample against number of training samples in a dataset that have been historically classified into four categories: judo players, jockeys, basket ball players and rugby players. Each of these categories is a pattern class that is represented by n-dimensional numeric attributes. Each training sample in the historical dataset is represented as a feature vector which is identified by the height and weight features. The first column in the dataset is the weight feature, the second one is the height feature and the third one is the class label which represents the output vector. The jockeys' category is identified by the class label 1, the basket ball players' category is identified by the class label 2, the rugby players' category is identified by the class label 3 and the judo players' category is identified by the class label 4. This dataset could be found in "Appendix A".

As the introduced classification problem is represented by two features only, the problem is a two-dimension classification problem that could be represented in a scatter diagram (see figure 3). The scatter diagram is only available in the 2-D classification problems. In case of having a multi-dimensions classification problems the visualization techniques will not be available easily, so that an eye assumption will be hard to be implemented. In our sports players' problem we can assume the class label of the unknown sample using the scatter diagram based on our eyes. For some multi-dimensions classification problems visualization by a sphere could be implemented. But the more dimensions the classification problem has, the harder the visualization is. Although this topic is beyond the scope of this thesis, it could be a base for future work.

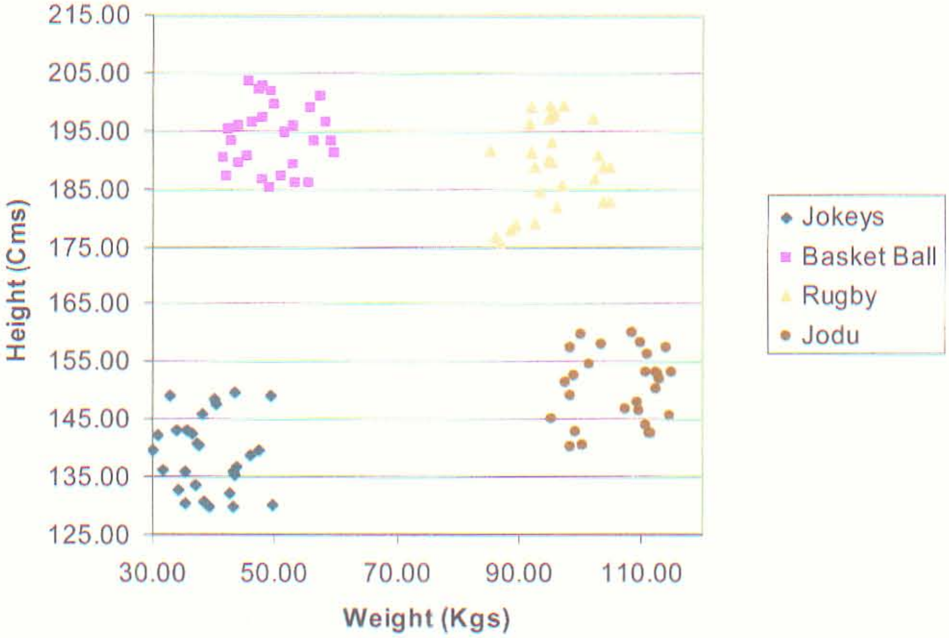


Figure 3. Features Represented in a Scatter Diagram

Now if we have the weight and height of a new sports player, we will be able to classify him to belong to one of the predefined classes we have in the dataset. We suppose that the new feature vector is identified as the following:

$$\underline{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 70 \\ 175 \end{bmatrix}$$

where x_1 is the weight feature and x_2 is the height feature. If we plot the new sample in the scatter diagram, we will notice that it is more close to the rugby and basket balls categories. But this assumption is not enough so we have to measure the distance between the unknown sample and all the training samples in order to search for the k training samples that are closest to the unknown sample. In this thesis MATLAB is used to implement the logic of this classification problem as found in "Appendix B".

After running the code, we find in the workspace of MATLAB the variable "minDistance" which represents the minimum distance between the unknown sample and the closest training sample in the historical dataset. In Addition, we find the variable "classLabel" which represents the class label to which the unknown sample belongs. In our case the variable "minDistance" will have the value "17.4" and the variable "classLabel" will have the value "3". So, the new unknown sample belongs to the rugby category (see table 1).

TABLE 1

<u>Distances between the unknown sample and the training sample</u>

Nearest Neighbor Experiment

Jockey	Basket Ball	Rugby	Judo
73.78	29.16	17.40	51.43
66.11	49.83	42.64	50.50
52.10	33.51	22.99	61.26
56.89	31.85	39.61	53.53
65.96	25.53	49.21	61.74
65.30	40.29	26.99	65.49
60.90	27.91	38.21	45.81
75.25	30.93	44.17	66.37
66.34	38.68	45.90	65.02
77.20	31.05	51.51	66.83
71.70	37.93	48.45	74.17
66.44	48.80	32.86	67.15
74.50	26.52	48.56	51.14
72.22	45.41	37.73	64.72
70.14	37.80	39.58	62.59
75.64	31.17	32.84	59.97
46.54	45.22	21.39	73.86
67.91	40.31	47.91	56.79
66.94	33.00	31.83	63.23
77.83	49.59	43.37	64.19
57.86	44.64	46.87	65.85
60.40	40.59	17.41	73.58
75.81	43.89	53.96	45.32
64.62	38.13	47.46	51.96
79.33	52.94	42.65	71.80
56.52	44.47	26.42	68.04
66.82	47.43	48.93	55.03
63.11	47.70	41.34	54.21

Performance Evaluation

In any classification problem the quality of the training samples play a very important role in evaluating the performance of the classifier used to solve the problem. So, if we want to have a better performance we have to undergo some preprocessing steps that may be applied to the data in the dataset in order to help in improving the accuracy, efficiency and scalability of the classification process. The importance of data preprocessing will be clearly identified in the following sections of this thesis where I will present the neural networks techniques. What is important in this phase where I have applied the Nearest Neighbor technique is the distribution of the training samples. If the training samples are clearly separable, the Nearest Neighbor classifier will match the unknown samples to their closest class in an efficient way. Where if the training samples are nor clearly separable and the classification problem is complicated, the Nearest Neighbor classifier will face hard times to classify the new unknown samples (see figure 4).

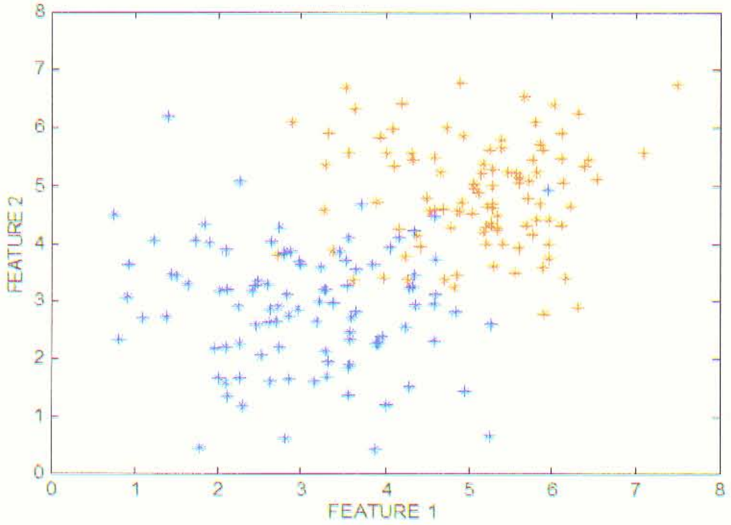


Figure 4. Complex Classification Problem Using the Nearest Neighbor Method

In figure 4 we can see that the problem is complicated where the samples of both classes (the red and the blue) are mixed up. So the unknown sample could have two training samples that have the same distance between each one of them and the unknown sample. But what if we have more than 2 class labels and the same problem occurs? As most of the real life classification problems are complicated and the samples are mixed up, we should have some sort of logic that will solve the problem in case of using the Nearest Neighbor classifier. This could be through setting a rule while applying the algorithm, that is, if such a case occurs, we can classify the unknown sample to any one of the available classes.

The Nearest Neighbor classifier is considered as a lazy learner because it needs to store all of the training samples and does not build a classifier until a new unknown sample is available for classification. With each new unknown sample the Nearest Neighbor classifier is measuring the distance between the unknown samples and all of the training samples, so that this technique is slowing the speed of recognition or learning. In much complex classification problems we will notice that the performance of the process is very slow. The Nearest Neighbor technique can experience expensive computational costs in the case of having a great number of historical data samples with which we need to compare an unknown sample. In addition, the Nearest Neighbor classifier assign equals weight to each feature in the dataset in that this may cause confusion in the case of having some redundancy or irrelevant features. So, the more efficient data preprocessing is the more accurate the classification result is.

Despite of the disadvantages of the Nearest Neighbor technique mentioned above, it has been already tested that this technique is the most accurate classifier among all other techniques. As the core logic in the Nearest Neighbor technique is measuring the distance between the unknown sample and all of the training samples, it results in at least one minimum distance. So, the probability of error in the Nearest Neighbor technique is less than the probability of error of any other classification technique.

Artificial Neural Networks - Single-Layer Perceptron Classifiers

We have seen in the previous chapter that although the Nearest Neighbor technique produces accurate classification results, it has some disadvantages that have led to the fact that the Nearest Neighbor technique is not the perfect solution for all classification problems. So, we have to ask: is there a better way to classify data?

Introduction to Linear Classifier

In order to prove that there is a better way for solving classification problems I introduce here a supposed 2-D problem that has only two features (weight & height) as the input vector. A linear classifier is used to classify the data samples into two classes (jockeys & rugby players) (see figure 5).

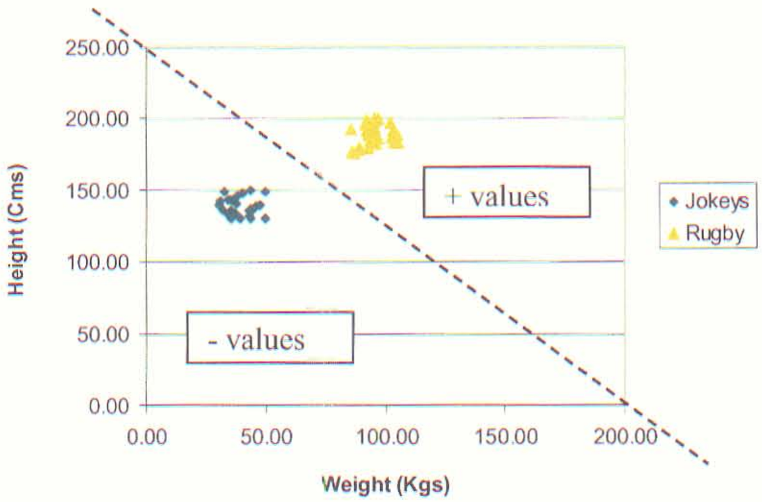


Figure 5. Linear Classifier

So, by using figure 5 we can simply say that on one side we have jockeys where on the other side we have rugby players. The classifier we have here is a straight line that can be obtained using the mathematical equitation of the straight line as follows:

$$y = mx + c$$ (where m is the slop and c is the part cut from the y axis)

$$y = -(250/200)x + 250$$

$$y = -1.25x + 250$$

$$y + 1.25x - 250 = 0$$

As a standard notation: y could be mapped to x_2 and x could be mapped to x_1, where x_1 and x_2 represent the features in the feature vector $\underline{X}$. So, the general form of the linear classifier is represented in the following function:

$$f(\underline{x}) = f(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_o = 0$$

By using the general equation of the straight line we can get w_0, w_1 and w_2

$$f(\underline{x}) = f(x_1, x_2) = 1.25x_1 + x_2 - 250 = 0$$

$$w_o = -250,\ w_1 = 1.25,\ w_2 = 1$$

The function's result will be zero, if we use any point that is located exactly on the line. In addition the function will have a positive value on one side and a negative value on the other side. So in order to prove this we need to use three samples to classify them: (80,150), (100,150) and (50,100)

$$f(\underline{x}) = f(x_1, x_2) = (1.25 * 80) + 150 - 250 = 0$$

$$f(\underline{x}) = f(x_1, x_2) = (1.25 * 100) + 150 - 250 = 25$$

$$f(\underline{x}) = f(x_1, x_2) = (1.25 * 50) + 100 - 250 = -87.5$$

From the results calculated above we can see that the first sample (80, 150) is a neutral sample as it is located exactly on the straight line. It could be classified as a rugby player or as a jockey. The second sample (100, 150) has a positive value, so that it is classified as a rugby player, whereas the third sample (50, 100) is classified as a jockey because it results in a negative value. So, as a general logic I could say that all the positive values are classified as rugby players and all negative values are classified as jockeys. I also could say that the negative values are rugby players and the positive values are jockeys if I multiply in minus values in the above three equations as follows:

$$f(\underline{x}) = f(x_1, x_2) = -(1.25 * 80) - 150 + 250 = 0$$

$$f(\underline{x}) = f(x_1, x_2) = -(1.25 * 100) - 150 + 250 = -25$$

$$f(\underline{x}) = f(x_1, x_2) = -(1.25 * 50) - 100 + 250 = 87.5$$

But is there only one classifier that could solve the problem we have? Actually there is infinite number of classifiers which can solve the given problem (see figure 6). Each one of these classifiers has its own weights.

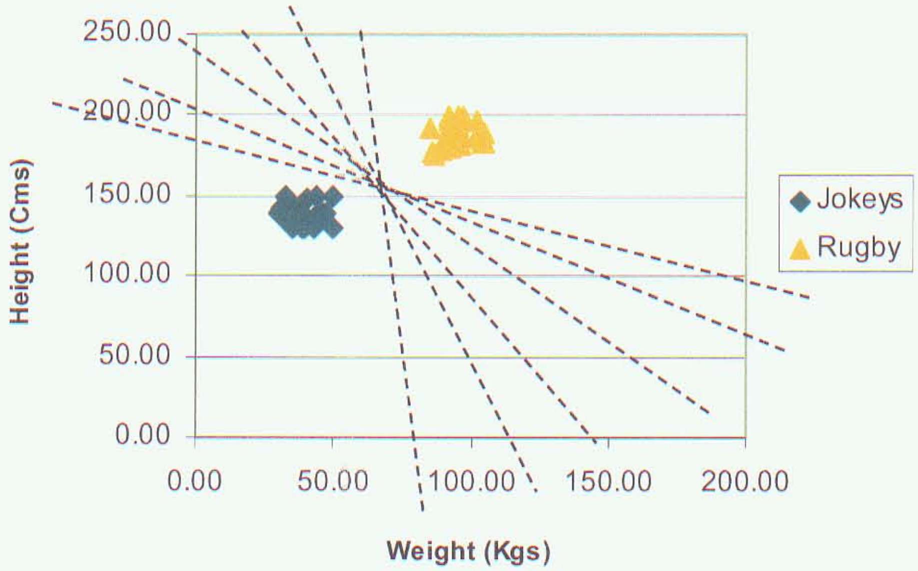


Figure 6. Infinite number of Linear Classifiers

Modeling of the Linear Classifier

The simple classification problem presented in the previous section can be modeled using a neural network technique that reflects the way in which the real neuron in the human brain operates (see figure 7).

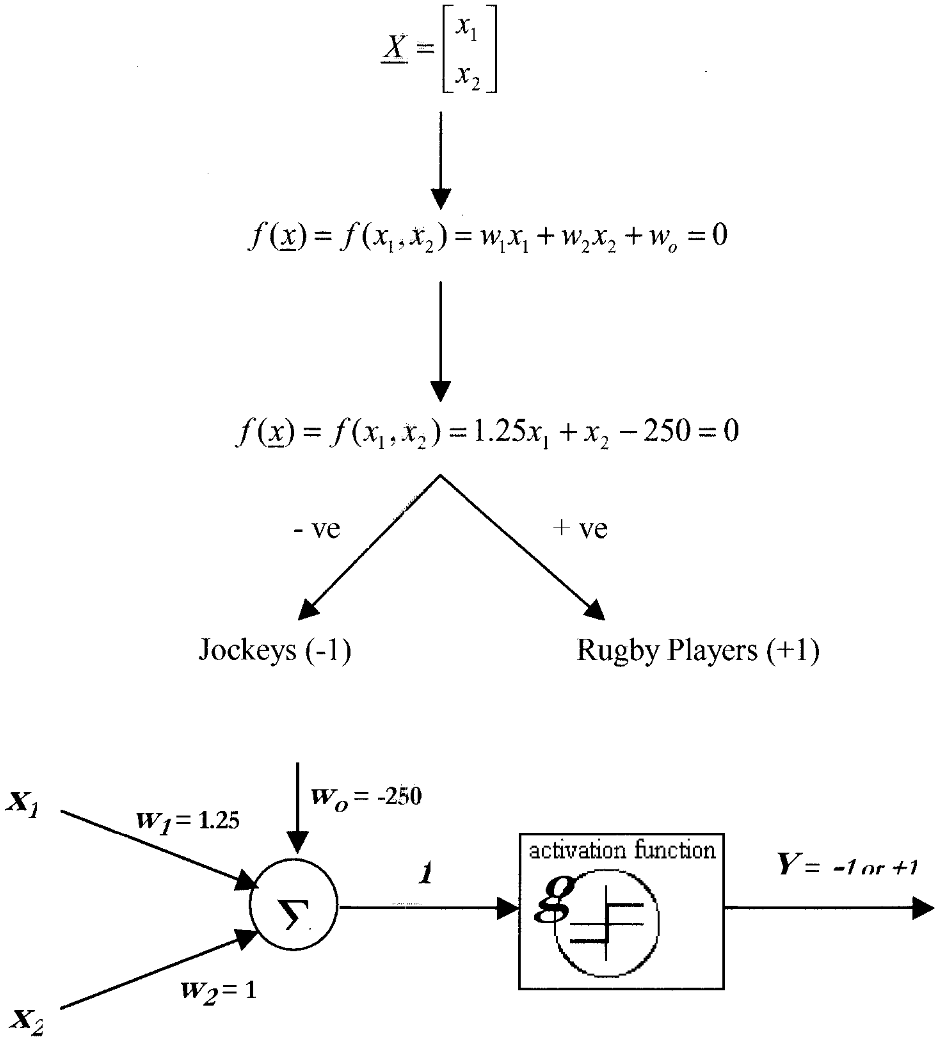

$\underline{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$
$f(\underline{x}) = f(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_o = 0$
$f(\underline{x}) = f(x_1, x_2) = 1.25x_1 + x_2 - 250 = 0$
- ve
+ ve
Jockeys (-1)
Rugby Players (+1)
X_1
$W_1 = 1.25$
$W_o = -250$
Σ
1
activation function
g
$Y = -1$ or $+1$
$W_2 = 1$
X_2

Figure 7. Modeling of the Linear Classifier

So, we can say that the linear classifier could be modeled by a simple artificial neuron. I will discuss this model in details in later in this thesis when reaching the single-layer perceptron and the multi-layer perceptron algorithms.

Architecture of the Single-Layer Perceptron

The linear classifier discussed in the previous 2 sections was the backbone of the perceptron classifier which represents the first-working artificial neural networks technique. In 1958 Rosenblatt invented the perceptron algorithm to be a simplified model of the biological neuron system. This algorithm is about learning the weights automatically from the given training data samples. By using figure 7 in the previous section we can now generalize the model to be suitable for classification problems that have *n* number of inputs or features (see figure 8).

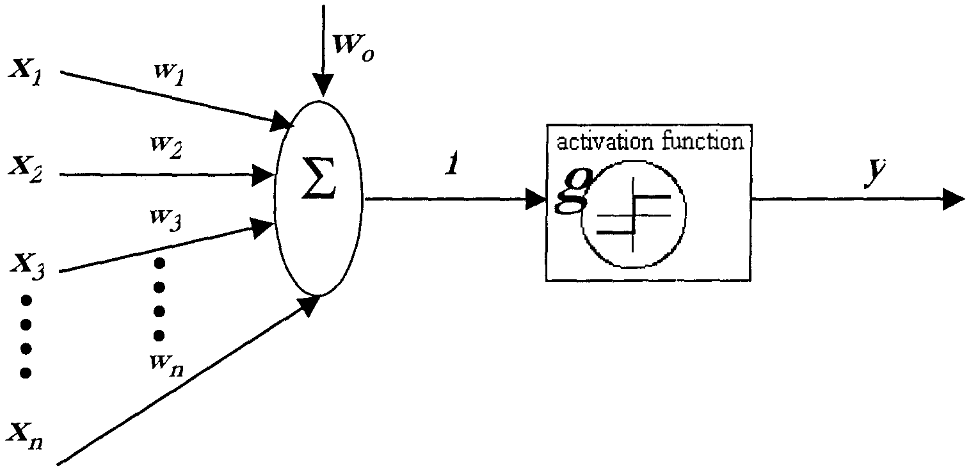


Figure 8. Generalized Model of the Perceptron Learning Algorithm

In figure 8 $x_1, x_2, x_3, \ldots, x_n$ are the *n* inputs to the artificial neuron, where w_1, w_2, $w_3, \ldots, w_n$.

As the biological neuron in the human brain receives all inputs and signals through dendrites, sums them and produces an output only in the case when the sum is greater than a threshold value, the input signals are passed on to the cell body through the synapses. The synapses could accelerate or hold back any signal. The process of acceleration of holding back the arriving input signals is the process that is modeled by the weights in the perceptron technique. The stronger the synapses are, the larger weights they have and vise versa. So, this could be reflected in the perceptron algorithm: the more important and relevant the features are, the larger weights they have in the training process.

In real life classification problems the perceptron is modeled as a single-layer feedforward networks that consists of many artificial neurons which is modeled in figure 8. In this type of networks there are two layers, namely the input layer and the output layer. The neurons of the input layer receive the input values whereas the output layer produces the output values. The input layer represents the feature space which contains the feature vector and the output layer represents the output space. Each input neuron in the input layer is connected to each output neuron in the output layer through the synaptic links which hold the weights. In any neural networks technique the input layer is not counted towards the total number of layers in the architecture. So as in the perceptron networks the architecture has an input layer and an output layer, the network is termed single-layer since it is the output layer (see figure 9).

The algorithm of the perceptron belongs to the supervised learning classifiers because the output classes are known and the weights are adjusted to minimize error whenever the actual output does not match the desired output. In the unsupervised learning the target output is not known, so the algorithm learns of its own by discovering the distribution of the features in the feature space. The unsupervised learning is beyond the scope of this thesis.

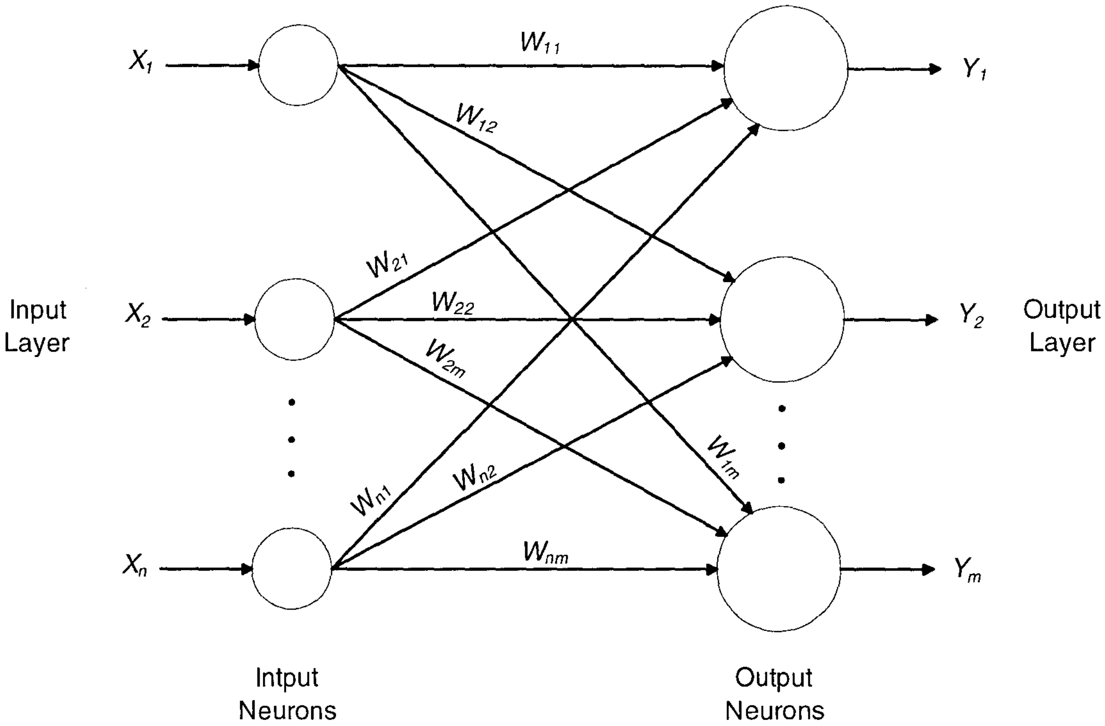


Figure 9. The Perceptron (Single-Layer Feedforward Network)

The Perceptron Algorithm

The perceptron algorithm aims to find the equation of the straight line that classifies the given data samples in the dataset. Once the slop and the part cut from the y axis is computed, the weights could be computed automatically. The following steps clarify the process of the perceptron algorithm:

- Given a dataset that contains the features (inputs) and a column for outputs
- Create a perceptron with (n+1) input neurons that are represented as a feature vector $\underline{x}$ $\{ x_0, x_1, x_2, x_3, \ldots, x_n \}$ where n is the number of inputs and x_0 is the bias input for each training data sample.
- Initialize the weight vector $\underline{w}$ $\{w_0, w_1, w_2, w_3, \ldots, w_n \}$ randomly.

- Iterate through the inputs of each data sample using their dedicated weights from the weight vector $\underline{w}$ in order to compute the weighted sum of the inputs for each data sample using the following equation:

$$(w_o + \sum_{i=1}^{n} w_i x_i)$$

- Compute the actual output Y_a using the activation function:

$$y_a = g(f) = g(w_o + \sum_{i=1}^{n} w_i x_i)$$

- Compare the actual output Y_a with the desired output Y_d for each data sample using the following equation:

$$\delta = y_d - y_a$$

  This equation computes the error initiated between the actual output and the desired output

- If all the data samples have been classified correctly where the actual output equals the desired output, output the weights and exit
- Otherwise, update the weights using the following equations:

$$\Delta w_i = \beta \delta x_i \text{ (Where } \beta \text{ is the learning rate and is a constant)}$$

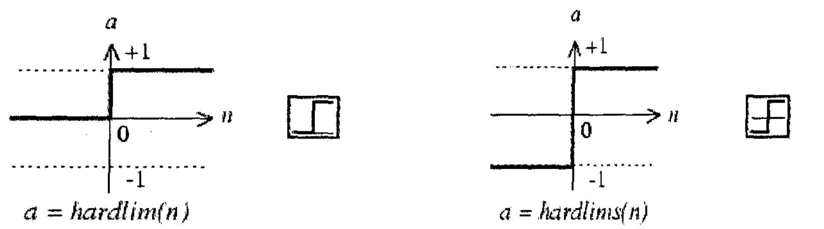
$$w_i(new) = \Delta w_i + w_i(old)$$

- Start again from step four with the new weights.

From the steps mentioned above we can figure out that by updating the weights on the links between the neurons of both layers, namely the input and output layers, the output values could be trained to match a desired output. Training is accomplished by sending a given set of inputs through the network and comparing the results with a set of target outputs. If the computed actual output does not match the desired output, the weights are adjusted to produce the closest output values. The updated weights are identified by adding an error correction value to the old weight. The training process is repeated until the performance of the network reaches the maximum rate of improvement (convergence). In this case the network is converged and the accurate weights are learned successfully. Then new testing data samples could be passed on to the network to be classified correctly.

During the training process the perceptron algorithm computes the actual output Y_a by using an activation function to be compared with the desired output Y_d. There are four types of the activation functions: hard-limit transfer function, symmetric hard-limit transfer function, log-sigmoid transfer function and tan-sigmoid transfer function. It is commonly known that the perceptron uses the hard-limit transfer function or the symmetric hard-limit transfer function. In hard-limit transfer function if the computed value is greater than 0, then the actual output Y_a is 1 else it is 0. In the symmetric hard-limit transfer function if the computed value is positive, then the actual output Y_a +1. But if it is negative, then the actual output Y_a is -1 (see figure 10).

The most widely process in applying the perceptron algorithm is to run the algorithm repeatedly through the training dataset until it finds a prediction vector (linear classifier) which classifies all of the training data samples correctly. This prediction rule is then used for classifying the testing dataset.



$a = hardlim(n)$
Hard-Limit Transfer Function

$a = hardlims(n)$
Symmetric Hard-Limit Trans. Funct.

Figure 10. Activation Functions for the Perceptron Learning Algorithm

Practical Experiments

In this section I will introduce two examples: the first one will be a very simple example with computations by hand and the second one is implemented in MATLAB. I have decided to follow this way in order to make the process of the perceptron clear and directly to the point.

Case Study: Computations by Hand

The classification problem that needs to be solved in this example is represented in the following table (see table 2):

TABLE 2

Dataset for the First Example for the Perceptron Algorithm

Feature 1 (x_1)	Feature 2 (x_2)	Class Label	Desired Output
1	2	A	+1
-1	2	B	-1
0	-1	B	-1

We need to find the right perceptron that will classify the given samples into different classes, namely class A and class B. So, I will start with a random classifier by which I can then compute the initially random weights, then the process should continue until reaching the convergence phase (see figure 11). In addition I will set the learning rate to be $\beta = 0.2$

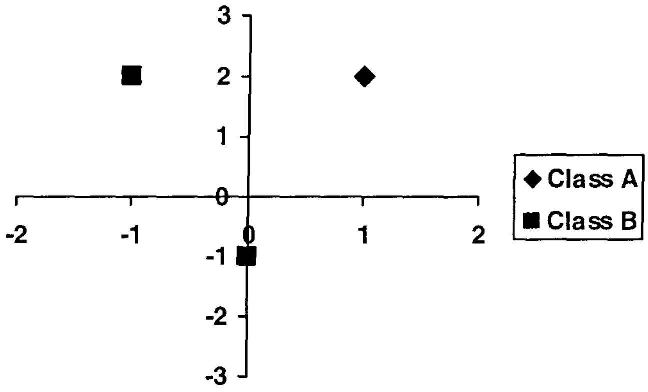


Figure 11. Scatter Diagram of the First Experiment for the Perceptron Algorithm

To show the target of this example I use here two iterations only as found in "Appendix C". The computations in the previous example should be repeated until the function (δ = Yd – Ya) evaluates to 0 for each given samples in the same iteration. We have also to note that I have used the symmetric hard-limit transfer function as the activation function because the desired output is +1 or -1.

Practical Experiment: MATLAB Code

In our example, the target is to find the suitable perceptron that will classify the given training data samples. This classification problem is similar to the one solved in the nearest neighbor classifier chapter, but this has only two class labels: jockeys and rugby players. Each training sample in the historical dataset is represented as a feature vector which is identified by the height and weight features. The first column in the dataset is the weight feature, the second one is the height feature and the third one is the class label which represents the output vector. The jockeys' category is identified by the class label -

1 whereas the rugby players' category is identified by the class label +1. This dataset could be found in the "Appendix D" of this thesis.

We could see that the output values are +1 and -1, so the symmetric hard-limit transfer function will be used as an activation function during applying the perceptron algorithm to solve this classification problem. If the desired outputs were 1 and 0, we can then use the hard-limit transfer function instead. As the introduced problem has only two input features, it is identified as a 2-D problem that could be represented in a scatter diagram (see figure 12). This example we have is just a simple classification problem that will reflect the way the perceptron algorithm could be implemented. In complex problems we will need to implement a more reliable solution.

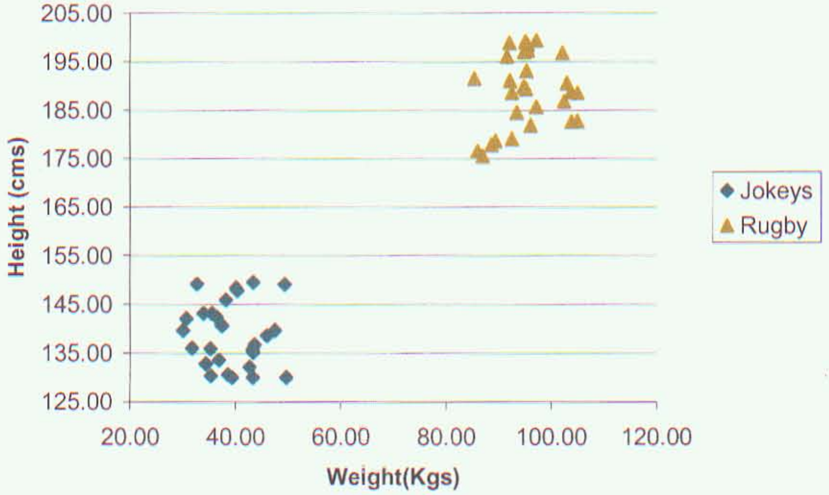


Figure 12. Scatter Diagram of the Second Experiment for the Perceptron Learning Algorithm

The introduced problem seems to be a linearly separable. So in this case the perceptron algorithm will make a finite number of mistakes and will converge to a straight line which correctly classifies all of the training samples. The MATLAB code shows this fact as found in "Appendix E".

Once the code is run, MATLAB plots the data samples and the perceptron which classifies the training data samples correctly (see figure 13). The convergence of the perceptron happens after finite numbers of errors.

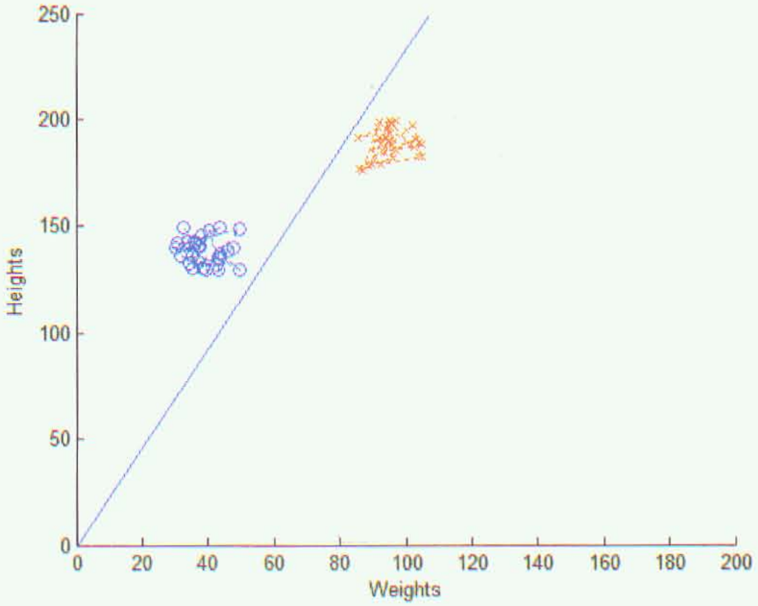


Figure 13. The Perceptron Classifies the Training Data Samples

Performance Evaluation

We have seen in the previous sections how the perceptron technique is better than the nearest neighbor technique. We have also seen that the perceptron converges with finite number of errors. But the question that should arise here: does it converge in all classification problems?

Actually to answer this question we have to use various classification problems. We suppose that we have the following classification problem (see figure 14).

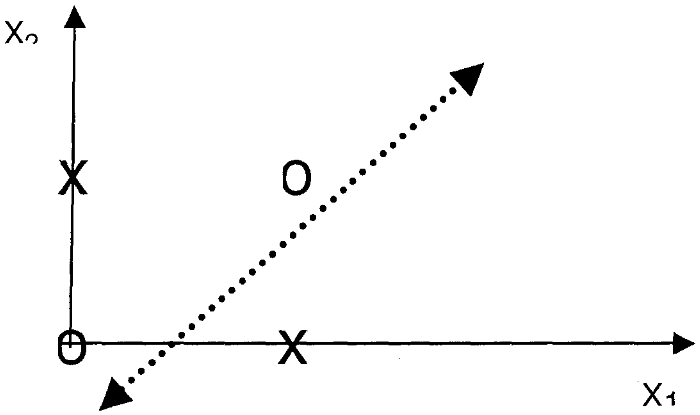


Figure 14. The XOR Problem

The problem has two classes, namely x and o. This problem is called XOR problem in field of data classification. It could be described by its truth table presented in (Table 3). We need to classify the inputs into two different classes: +1 and -1. The perceptron drawn in figure 14 could not solve the problem at all.

TABLE 3

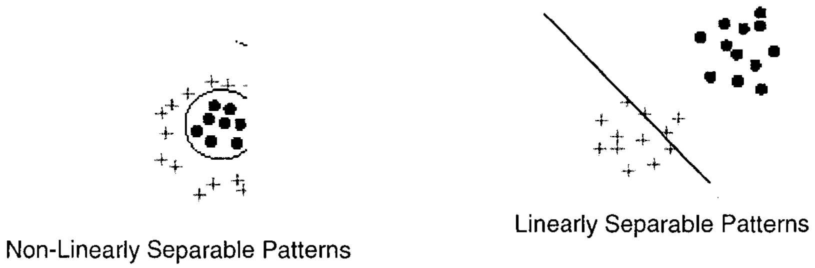
XOR Problem Truth Table

X_1	X_2	Output
0	0	-1 (class o)
1	1	-1 (class o)
0	1	+1 (class x)
1	0	+1 (class x)

So solving this problem by using a single perceptron is impossible. The XOR problem delayed research in the artificial neural networks field for about ten years, because it destroyed the perceptron technique invented by Rosenblatt. This failure was the motive for finding a better solution to reach to a standard in solving classification problems using neural networks. As a result, the XOR problem could be solved by using two perceptrons.

After extensive experiments, the perceptron convergence theorem was introduced: the process which uses the perceptron algorithm will only make a finite number of errors, if and only if all training samples are linearly separable. This theorem could be proved by measuring the distance between the currently maintained weight vector and the target weight vector. When the distance becomes smaller after updating the weights, it proves the convergence theorem of the perceptron algorithm.

This means that the perceptron can't find weights for classification problems that are not linearly separable (see figure 15). In real life classification problems we can't know whether the training samples are linearly separable or not. As a result it was discovered that the single-layer perceptron is not an efficient technique for classification.



Non-Linearly Separable Patterns

Linearly Separable Patterns

Figure 15. The Perceptron with Linearly and Non-Linearly Separable Data

The researches were targeting to find a more efficient classification technique that could solve the complex problems in the real world. In the next chapter I introduce the most important technique that has played a very efficient role in the field of neural networks and that has helped in solving complex classification problems: the multi-layer perceptron technique.

Artificial Neural Networks - Multi-Layer Perceptron Classifiers

In the single-layer perceptron we have seen that there are only two layers, namely the input layer and the output layer. In this chapter I discuss the most well-known and efficient neural networks architecture for classification which consists of more than one layer. The new layers added to this network architecture are hidden layers that reside in the intermediary section between the input layer and the output layer. This architecture is a generic neural network framework that could be applied in all supervised learning problems.

Architecture of the Multi-Layer Perceptron

The multi-layer perceptron is called multi-layer feedforward network, because it is made up of multiple layers, namely the input layer, the output layer and the hidden layers (see figure 16). The hidden layer helps in processing useful intermediary computations before outputting the final result. The input layer neurons are connected to the hidden layer neurons and the weights of the perceptrons linking the input layer and the hidden layer are referred to as input-hidden layer weights. On the other hand, the hidden layer neurons are connected to the output layer neurons and the weights both layers are referred to as hidden-output layer weights. In some networks there could be more than one hidden layer. It depends totally on the complexity of the given classification problem. After deep studies in this field it has been proven that two hidden layers with large numbers of neurons are sufficient to solve any classification problem.

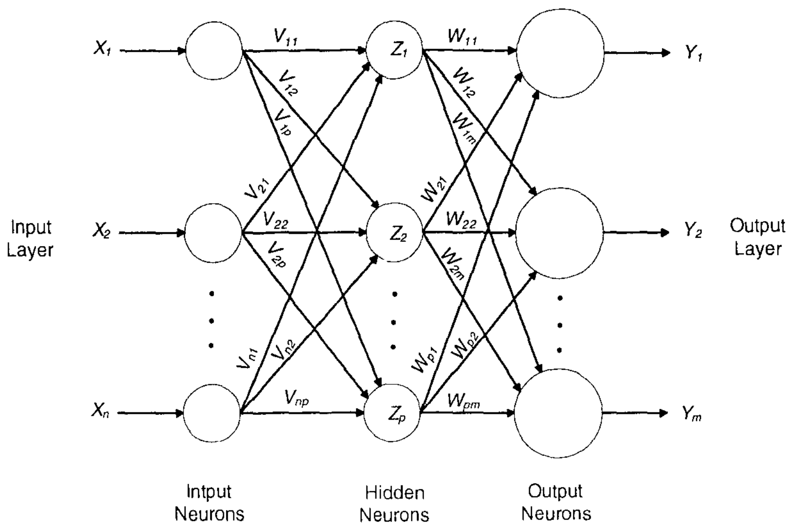


Figure 16. The Multi-Layer Perceptron Model

The architecture of the multi-layer perceptron is some how similar to the architecture of the single-layer perceptron. The inputs represent the features measured for each training sample and are fed concurrently into the input neurons of the input layer. The computed outputs of these neurons are, in turn, fed concurrently into the neurons of a second layer, known as a hidden layer. The computed outputs of the hidden layer could be treated as the inputs for another hidden layer and so on. The computed outputs of the last hidden layer are then fed simultaneously to the neurons of the output layer, which compute the final actual outputs for the given samples.

Designing a Multi-Layer Perceptron Network Topology

When constructing a network that is based on multi-layer perceptron architecture, we should first decide on the network topology by determining the number of neurons in the input layer, the number of hidden layers, the number of neurons in each hidden layer and the number of neurons in the output layer. The number of input neurons represents the number of features in the training dataset, whereas the number of the input neurons represents the number of class labels defined as the desired output in the training dataset.

The network design is a trial-and error process and may have a great influence on the accuracy rate of the trained results. The randomly initialized weights and the bias values for each neuron may also affect the resulting accuracy. If the resulting accuracy is not acceptable, it is common to repeat the training process with different weights or/and different topology in order to reach the desired output values.

The XOR Problem with Multi-Layer Perceptron

I have introduced the XOR problem in section 3.6 with an initial approach to solve this linearly inseparable problem. This approach was about having more than one perceptron; each one classifies small linearly separable sections of the training inputs. The next step is to combine their computed output into a new perceptron which will results in a final classification technique that identifies the class to which each input belongs (see figure 17).

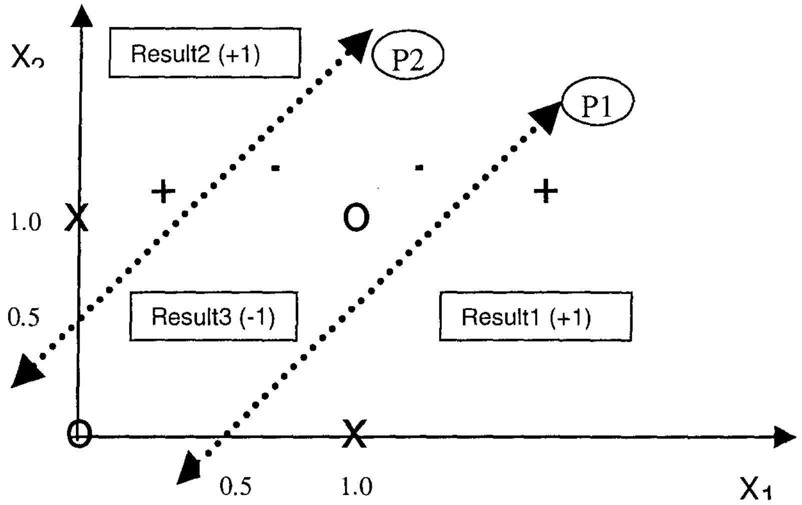


Figure 17. The Solution for the XOR Problem

I assume here that the perceptron P1 cuts the x_1 axis at the point 0.5 and cuts the x_2 axis at the point -0.5. The perceptron P2 cuts the x_2 axis at the point 0.5 and cuts the x_1 axis at the points -0.5. As we have known how to get the weights for any perceptron, we can get the weights for P1 and P2 as follows:

Weights for P1:

$$y = mx + c$$

$$y = (0.5/0.5)x + (-0.5)$$

$$y = 1x - 0.5$$

$$y - 1x + 0.5 = 0$$

$$f(\underline{x}) = f(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_o = 0$$

$$f(\underline{x}) = f(x_1, x_2) = -1x_1 + 1x_2 + 0.5 = 0$$

$$f(\underline{x}) = f(x_1, x_2) = 1x_1 - 1x_2 - 0.5 = 0$$

Weights for P2:

$$y = mx + c$$

$$y = (0.5/0.5)x + 0.5$$

$$y = 1x + 0.5$$

$$y - 1x - 0.5 = 0$$

$$f(\underline{x}) = f(x_1, x_2) = w_1x_1 + w_2x_2 + w_o = 0$$

$$f(\underline{x}) = f(x_1, x_2) = -1x_1 + 1x_2 - 0.5 = 0$$

The weights for perceptron P1 are: w_0 =- 0.5, w_1 = 1, w_2 = -1.

The weights for perceptron P2 are: w_0 = -0.5, w_1 = -1, w_2 = 1.

Please note that I have multiplied the equation for the perceptron P2 in -1 to get the correct output values. We can then use the computed weights to get three results (see table 4):

Result1: P1 is positive and P2 is negative

Result2: P1 is negative and P2 is positive

Result3: P1 is negative and P2 is negative

TABLE 4

Truth table of the Desired Outputs for the XOR Problem

P1	P2	Computed Output
+1	-1	+1 (class x)
-1	+1	+1 (class x)
-1	-1	-1 (class o)

The first two results classify the data samples into the class label +1 and the third result classifies the data samples into the class label -1. The XOR problem could be modeled using a neural network topology that consists of an input layer, a hidden layer and an output layer (see figure 18). For the perceptron of the hidden layer, the inputs come from the features of the XOR problem. But for the perceptron of the output layer, the inputs are the outputs of the hidden layer. The values specified in table 4 represent the outputs from the input neurons and in the same time represent the input for the output neuron. The weights of the perceptron resulted from combining the perceptron P1 and the perceptron P2 are as follows:

$w_0 = 1, w_1 = 1, w_2 = 1.$

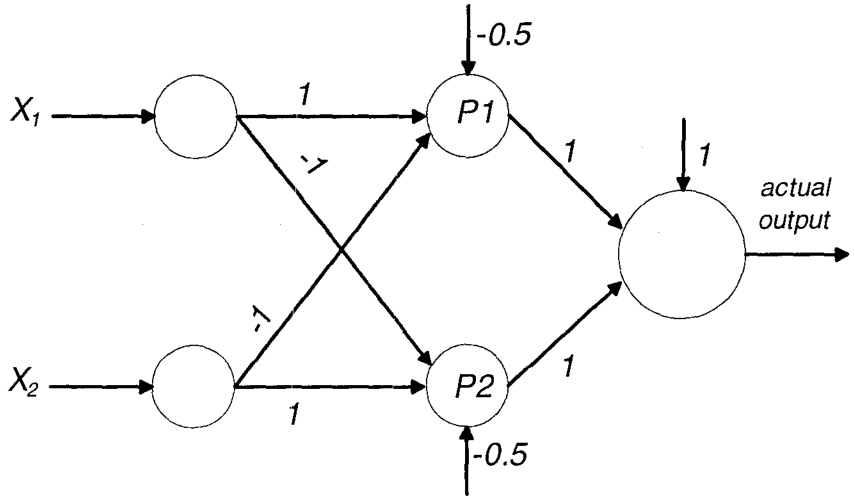
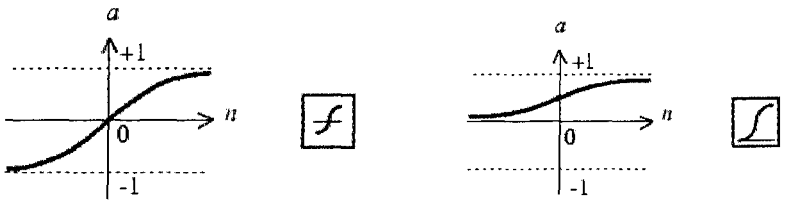


Figure 18. XOR Problem Model with MLP

So, the neurons in the output layer does not know the actual input values because of the hidden layer which acts as a masking layer between the input layer and the output layer. In the single-layer perceptron technique we could use the hard-limit transfer function or the symmetric hard-limit transfer function as an activation function while running the perceptron algorithm. In the XOR problem which represents the multi-layer perceptron technique, these transfer functions won't give us any indication of the scale by which the weights of the network could be adjusted. So, the other two types of the activation function should be used here in order to smooth the results. These new activation functions are: the log-sigmoid transfer function and the tan-sigmoid transfer function (see figure 19). The sigmoid function is non-linear and helps in modeling linearly inseparable classification problems. This function is very useful when using the backpropagation algorithm.



$a = tansig(n)$

Tan-Sigmoid Transfer Function

$a = logsig(n)$

Log-Sigmoid Transfer Function

Figure 19. Activation Functions for the Multi-Layer Perceptron with Backpropagation

Backpropagation Concept

By using the multi-layer perceptron networks the weights that connect the last hidden layer and the output layer are the first weights that are updated. Because the multi-layer perceptron technique contains hidden layers between the input layer and the output layer, all the weights that connects the input layer and the hidden layer needs to be updated, if the weights does not converge during the first epoch. The backpropagation technique is then used to perform this updating process.

The backpropagation network is a multi-layer perceptron network that consists of an input layer, one or more hidden layer and an output layer. The neurons in the network architecture are connected in a feed-forward way with input neurons connected to hidden neurons and hidden neurons connected to output neurons. When a backpropagation network is cycled, an input pattern is propagated forward to the output neurons through the intermediary neurons of the hidden layer(s).

The learning process in the backpropagation networks happens by processing the training data samples repeatedly and comparing the actual computed output for each sample with the desired output or the class label. If the computed output does not match with the actual output, an error signal is identified and is then propagated in the backwards direction from the output layer through the hidden layer(s) down to the input layer in order to update and adjust the weights in each layer of the network. From this process in the backwards direction, the name "backpropagation" has been developed. After finite number of iterations or epochs the network will converge, if and only if the training samples are processed to be clean, relevant to the classification problem and normalized.

Backpropagation Algorithm

The following steps illustrate the how the backpropagation algorithm, which is proposed by Rummelhart and McClelland, works:

- Normalize the inputs and outputs. Normalizing is about scaling the values for a given feature so that they fall within a small specified range, such as -1.0 to 1.0, or 0.0 to 1.0
- Assume the number of hidden layers and the number of hidden neurons

- Initialize the weights and the biases to small random numbers, usually ranging from -1.0 to 1.0 or from -0.5 to 0.5.
- Propagate the inputs forward by the following steps:
- feeding the training sample to the input layer
- Compute the inputs to each neuron in the first hidden layer as a liner combination using the function $(w_o + \sum_{i=1}^{n} w_i x_i)$ as done in the single-layer perceptron. These inputs represent the outputs of the neurons of the input layer.
- Compute the inputs to each neuron in the second hidden layer as specified in the previous step and so on till completing all the hidden layers specified in the second step.
- Evaluate the output of each neuron in the hidden layer by using the sigmoid function as an activation function. Two types of sigmoid functions could be used here, namely the log-sigmoid transfer function or the tan-sigmoid transfer function. The result of the sigmoid function represents the output of each neuron in the hidden layer.
- Compute the inputs to the output layer as specified in the hidden layer.
- Evaluate the output of each output neuron using the sigmoid function.
- Calculate the error and the difference between the actual output y_a and the desired output y_d for each output neuron by using the function $\delta_o = y_d - y_a$
- If the actual output matches with the desired output, repeat the steps with the next inputs in the training dataset. If the actual output does not match the desired output, backpropagate the error by the following steps:

- Calculate the error for all weights of each neuron in the output layer using the delta rule: $\Delta w_i = \beta\delta(Y_a(1-Y_a))x_i$ (where β is the learning rate and x_i is the input for the output neuron calculated by the sigmoid function)

- Compute the new weights in the output layer using the function:

  $w_i(new) = \Delta w_i + w_i(old)$

- Compute the error for each neuron in the last hidden layer using the function:

  $\delta_h = (\sum_{i=1}^{n} w_i(new)\delta_o)$ (where n is the number of weights for each hidden neuron and δ_o is error of the output neuron connected to the hidden neuron through the weighted link $w_i(new)$)

- Calculate the error for each weight in the last hidden layer that represents the weights between the last hidden layer and the previous hidden layer using the delta rule.
- Calculate the new weights that connect the last hidden layer and the previous hidden layer.
- Repeat the three previous steps to get the new updated weights for all assumed hidden layers and for the input layer.
- Repeat the process starting from the fourth step until the convergence in the error is less than the tolerance value.

When the network converges, the learning process is finished and we start working on the testing process.

The Effect of the Learning Rate β

The backpropagation learning algorithm aims to collapsing the network's structure to a single vector of weights, namely a single perceptron. During the process many factors affect the performance of the learning and the convergence rate. The learning rate is one of these factors. The learning rate determines the size of the weight updates during each iteration. Wrong choice of the learning rate can have a negative impact on the final result.

If the learning rate is lowered, the iterations needed to solve the problems will be increased and the time of convergence will be increased too. This will result in a slower learning process. But on the other hand if the learning rate is increased, the needed iterations will be decreased significantly, thus resulting in a faster learning process. As the learning rate is used in the delta rule, the weights will be affected by changing it. If we use a very low rate, the weights will be very small, but the in this case we will assure to reach to convergence even after spending a lot of time. In the case of using a very big rate, the weights become very big. This could result in a failure in the convergence.
So, the choice of the learning rate is a very tricky task in backpropagation algorithm. The range of learning rate that produces a faster training depends totally of the criteria of selecting the features and their number. Eaton and Oliver (1992) have suggested an empirical formula to select learning rate as follows:

$$\beta = \frac{1.5}{(N_1^2 + N_2^2 + \cdots + N_m^2)}$$

where N_1 is the number of patterns of type 1, N_2 is the number of patterns of type 2 and m is the number of different pattern types.

The researches for improving the learning performance of the backpropagation algorithm focus mainly on the gradient-based algorithms with adaptive learning rate. There are many strategies that help in selecting the best learning rate. The following are some of them:

- Start with a small learning rate and increase it, if successive iterations decrease the error rate, or decrease it, the error rate increases significantly.
- Start with a small learning rate and increase it, if successive iterations keep the gradient direction fairly constant, or decrease it, if the gradient direction varies during each iteration significantly.
- Set a unique learning rate for each weight. It should be increased, if the weights are changing successively in the same direction.
- Use a formula to calculate the learning rate, such as the one mentioned above which introduced by Eaton and Oliver (1992).
- Set a learning rate to $1/t$, where t is the number of iterations through the training set so far.

The aim of all of the above-mentioned strategies is to decrease the convergence error during iterations, to secure the ideal convergence for the training algorithm, and to avoid getting stuck at a point where the weights appear to converge, but are not the ideal solution.

If we have more efficient information about the input patterns, this could lead to a better selection of the learning rate. If the learning rate is small, that is, less than 0.2, the weights are adjusted in small increments. In this case the network converges more slowly but with little oscillations because the step length will be short. On the other hand if the learning rate is large, that is, greater than 0.5, the weights are adjusted radically, thus, resulting in missing the optimum combination of weights and in large oscillations as the step length will be longer. So the learning rate should be selected as large as possible in order to allow fast learning without oscillations.

There is no any empirical method that helps in selecting the best learning rate, but a trial-and-error process may lead to the best optimum value of learning rate. The learning rate varies depending upon the input patterns of the classification problem and the complexity of the problem. I could say that if the input patterns are mixed up in a drastically way, this may lead to a lower selection of the learning rate and vise-versa. But in classification problems that are represented by multi-dimensional input vectors, it could be difficult to identify the complexity of the problem. In most multi-dimensions cases the complexity rate is high, thus, resulting into more efforts to select the optimum value of the learning rate.

As the learning rate impacts the position of the weight vector, it can't be negative because this would make the weight vector to move away from the ideal one during the learning process. In addition, if the learning rate is zero, the learning process won't take place at all. As a result, the learning rate should always be positive. If the learning rate is equal to two then the network becomes unstable and if the learning rate is greater than one, the computed weight vector will move away from its optimum position, thus resulting in oscillation. So, the learning rate must be between zero and one. So the selection of the learning rate has a very big impact on the general performance of the learning process.

The Effect of the Hidden Nodes

The given classification problem decides the number of nodes in the input layer and in the output layer. The number of features is the same number of the input neurons, whereas the number of class labels represents the number of the output neurons. But how could we decide the number of hidden neurons?! In general there is no an empirical criterion about deciding the hidden neurons. As a starting guide, we can start with the minimum number of the hidden neurons. This would help in keeping the memory demand for storing the weights to the minimum. Then we may start to increase the number of the hidden neurons until to reach the desired convergence.

Mirchandani and Cao (1989) have found a relation between the number of hidden neurons in the backpropagation networks and the separable regions in the input space. They have proved this by the following function: $H = M - 1$, where M is the number of separable regions in the input space and H is the number of hidden neurons. Huang and Huang (1991) have introduced another way to decide the hidden neurons based on the number of elements in the training dataset. They have proved argued that the optimal number of hidden neurons is empirically found out to be $H = K - 1$, where K is the number of elements in the training dataset. For them this is the least upper bound on the number of hidden neurons needed to realize an arbitrary real valued function defined by set with K elements. So, many trials have been made to reach to kernel point, from which we can select the number of hidden nodes.

It has been experimentally proved that the learning process could be significantly faster when the number of the hidden neurons is equal to the number of input patterns. In this case the weight vectors related to each input and output pair can be combined. In the next section I will introduce a practical example that shows how the number of the hidden layers and the number of the hidden nodes has a great impact on the convergence rate and on the total performance of the network.

In general, the trial-and-error method is considered the better solution for selecting the number of hidden neurons. Depending on the complexity of the classification problem, we can then try different network topologies in order to reach to the most optimum network architecture. In most cases, increasing the number of the hidden neurons could improve the performance of the network on the training dataset, but not necessarily on testing dataset. If we add enough hidden units, then the network performance will be greatly improved, because it will have enough weights to exactly represent all the training patterns. This may result in creating a very poor network with a very slow convergence rate and with a little ability to generalize or classify the input patterns on the testing process.

The most powerful method, that helps in assessing the effect of the number of hidden neurons on any classification problem, is measure the performance and the convergence rate on the testing dataset. When the total number of the hidden neurons is increased, the network performance on the testing dataset improves significantly. This is because each new hidden unit will represent one of the input features. But we have to be aware of the most optimum number of the hidden neurons, as increasing the number to a large value will cause a decrease in the network performance and the network will lose the power of generalization which helps a lot in the case of high complexity. In this case the network starts to learn the noisy data in the dataset. I will discuss this in the next section when working on a practical example.

Practical Experiments

In this section I will introduce two examples: the first one will be a very simple example with computations by hand and the second one is implemented in MATLAB. I have decided to follow this way in order to make the process of the multi-layer perceptron with backpropagation clear and directly to the point.

Case Study: Computations by Hand

In this example I discuss here a simple classification problem that is represented in figure 20.

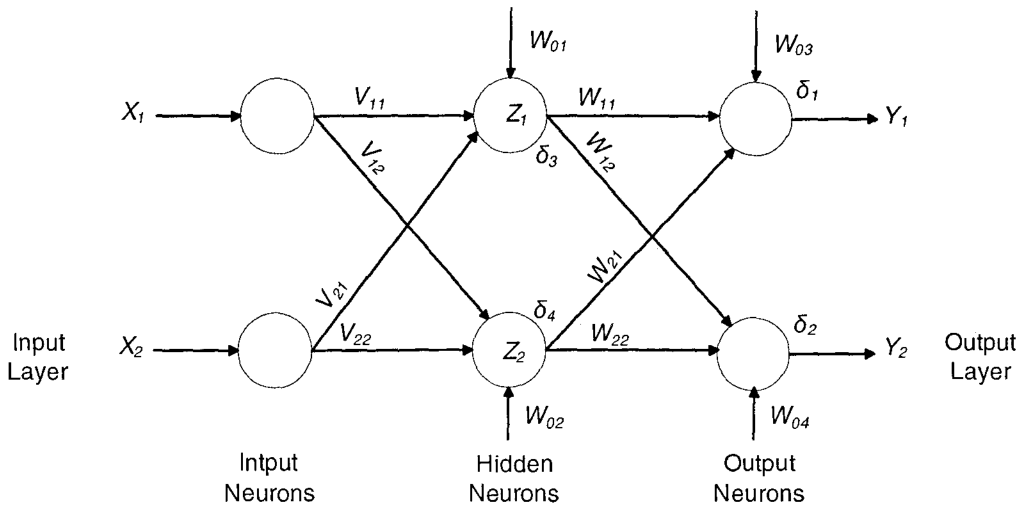


Figure 20. Model of the First Experiment for the MLP with Backpropagation Algorithm

The problem has two input neurons, two hidden neurons and two output neurons. The hidden neurons are represented in one hidden layer that is the intermediary layer between the input layer and the output layer. I aim here to show how the backpropagation algorithm operates in the case of multi-layer perceptron network. So, the target of the problem is to calculate some computations for the error in the output layer, the backpropagated error in the hidden layer and then the new adjusted weights for one iteration and for one input pattern only, as I don't target the convergence here.

The input values are:

- $X_1 = 1$
- $X_2 = 0.5$

The initial weights are:

- $V_{11} = 1$
- $V_{12} = 0.5$
- $V_{02} = -1.5$
- $V_{21} = -1$
- $V_{22} = -0.5$
- $W_{01} = 0$
- $W_{11} = 2$
- $W_{12} = 1.5$
- $W_{02} = -1.5$
- $W_{21} = 1$
- $W_{22} = -1$
- $W_{03} = 0$
- $W_{04} = -0.5$

The desired outputs are:

- $Y_1 = 0.8$
- $Y_2 = 0.2$

The learning rate is:

- $\beta = 0.3$

The required computations are:

- Find the new adjusted weights $V_{11}, V_{12}, V_{21}, V_{22}, W_{11}, W_{12}, W_{21}, W_{22}$
- Find the backpropagated errors δ_3 at node Z_1 and δ_4 at node Z_2

The solution could be found in "Appendix F".

These computations implemented in the example belong to the first epoch only. So, in order to reach to the convergence, I should continue with more epochs. In each epoch, I should follow the same logic as I did in the first one. The target is to adjust the weights until reaching the optimum weight vector which classifies the input data into two separate classes.

Practical Experiment: MATLAB Code

In this classification problem, the target is to find the optimum weight vectors that classify the given samples into five classes. There are two datasets available for this example, namely the training dataset and the testing dataset, could be found in "Appendix G". The major logic of the multi-layer perceptron technique with backpropagation will be implemented on the training dataset to let the system learn the optimum weights. Once the correct weights are learnt, the testing dataset will be processed to measure the performance of the computed weights.

The problem is about classifying all the samples in the dataset into five classes of cars, namely small cars, midsize cars, compact cars, large cars and sport cars. Each of the training dataset and the testing dataset contains 250 data samples and 15 columns. The first 14 columns are the input features which are already normalized. The given input features are: Price, City MPG, High Way MPG, # Cylinders, Engine Size, Horse Power, RPM at max HP, Revs/min, Fuel Tank Capacity, Passenger Capacity, Length, Width, Wheel Base, and Weight. All of these features describe the type of each car. The last column in the training dataset represents the class labels as follows: the small cars class is identified by the class label 1, the midsize cars class is identified by the class label 2, the compact cars class is identified by the class label 3, the large cars class is identified by the class label 4 and the sport cars class is identified by the class label 5. The class label column represents the desired output for each given data sample in the training dataset.

In the MATLAB code created for this problem a binary code has been used to represent the output classes as follows:

1. Small	=	[0 0 0 0 1]
2. Midsize	=	[0 0 0 1 0]
3. Compact	=	[0 0 1 0 0]
4. Large	=	[0 1 0 0 0]
5. Sporty	=	[1 0 0 0 0]

This binary code represents the desired output identified in the 15th column in the training dataset. The actual outputs for each data sample will be compared against its binary code. If they are equal to the binary code or close to it by a certain degree, the data sample will identified as a correct classified sample. On contrary, if the actual outputs are not close to the binary code, an error is then occurred and added to the error function that sums all the errors occurred during running the process.

As this problem has a multi-dimensions input vector and five class labels, it is most probably that the problem is non-separable and complex, that's why the MLP technique is used here. In this case the MLP with backpropagation algorithm may not make a finite number of errors. So, it is a must to create stopping criteria that would let the system stops processing, as in most of complex problems a hundred percent effective convergence is not guaranteed at all. Some commonly used stopping criteria are:

- stop after a certain number of runs through all the training data (each run through all the training data is called an epoch);

- stop when the total sum of the error reaches some low level.

Both approaches will be shown by the code created in MATLAB.

As the best network topology that would classify the given data samples correctly is not known, the solution of this problem will be starting from a single-layer, namely the output layer. If the error rate is above the accepted value, the topology will be adjusted to add a new hidden layer. And if the error rate is still high, the hidden nodes in the hidden layer will increased. The topology will be then adjusted to have two hidden layers, if the error rate is not lower than the tolerance value. It is like an adaptive neural network solution which is adjusted until finding the optimum weight vectors.

The solution of this problem will be implemented in different experiments. Each one of them represents a suggested network topology. In order to get the optimum topology, we have introduced some sort of logic in the MATLAB code which measures the error rate at the end of running the code, that is, after finishing the specified epochs. This logic is implemented by a function which compares the desired output with the actual output and calculates the error for each data sample in the training dataset. As this function will be implemented on the training dataset and on the testing dataset, it also gets the percentage of each dataset's accuracy. In all experiments we will run the same code more than once, as the weights are initialized randomly. This means that the learning accuracy may change at each time when we run the code. It is not guaranteed at all to have a high testing accuracy, if the training accuracy is already high. So, the best topology will be the one that achieves a high training and testing accuracy.

Experiment Sets

Experiment 1:

The network topology introduced in this experiment consists of the input layer and the output layer. As the training dataset and the testing dataset contain 14 input features for each, the input layer will always have 14 input neurons. In addition, the output layer will always have five output neurons, because the data samples are required to be classified into five classes of cars as mentioned above. This topology is the base that will be used in building other topologies in the coming experiments.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as in the following table.

TABLE 5

Results of the first experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
83.6%	41	84.4%	39	64%	90	64.8%	88

It is obvious that there is a slight difference in the accuracy between both trials. The most important observation in this experiment is the accuracy of the testing dataset. In both trials it was 64% and 64.8% respectively. This accuracy by any means is not acceptable at all. So the main core of solving this classification problem is to have a higher testing accuracy with lower number of errors. The number of errors occurred represents the number of data samples that are not correctly classified in respect to their desired outputs. It is noticeable during running the code that the learning process is quick enough, this because the network topology is not complex at all, as it is considered as a single-layer perceptron.

Experiment 2:

In this experiment the network topology will be based on the main topology that was created in the first experiment, but a new hidden layer will be added between the input layer and the output layer. The hidden layer will contain five hidden nodes.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as follows:

TABLE 6

Results of the second experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
85.6%	36	84%	40	66%	85	64%	90

It is clear that there is a small difference in the training accuracy for both trials. It is also noticeable that the testing accuracy is still not acceptable. To accept the testing accuracy, it should have a value near to the value of the training accuracy. During running the code, the learning process is a little bit slow, this is because the network topology is getting complex, as it consists now of two layers, namely the hidden layer and the output layer. In this topology the number of weights is distributed as follows: 70 weights that connect the input layer with the hidden layer with additional five biased weights for the five hidden nodes and 25 weights that connect the hidden layer with the output layer with additional five biased weights for the five output nodes. So the total number of weights is 105. The more weights the MLP network has, the more time it takes to learn the optimum weights. This fact will be very clear in the next experiments.

Experiment 3:

In this experiment the network topology will be the same as the one created in the second experiment, but a more five hidden nodes will be added in the hidden layer that is the intermediary between the input layer and the output layer. So, the hidden layer will contain ten hidden nodes instead of five hidden nodes that were specified in the second experiment.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as in the following table:

TABLE 7

Results of the third experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
98%	5	88%	30	84.8%	38	69.6%	76

The results of this experiment were totally astonishing. There was a big difference between the two trials of the training dataset: the training accuracy for the first trial was 98% and the number of error was only five data samples, but in the second trial the training accuracy was 88% and the number of errors was 30 data samples. This result proves that the weights, which are initialized randomly, play a very important role in determining the accuracy of the learning process. So, the testing accuracy in the second trial was also lower than in the first trial. In general there is still a big difference between the training accuracy and the testing accuracy. In this topology the number of weights is distributed as follows: 140 weights that connect the input layer with the hidden layer with additional ten biased weights for the ten hidden nodes and 50 weights that connect the hidden layer with the output layer with additional five biased weights for the five output nodes. The total number of weights is 210, so that the learning process takes more time and gets slower.

Experiment 4:

In this experiment the network topology will be the same as the one created in the third experiment, but a more five hidden nodes will be added in the hidden layer that is the intermediary between the input layer and the output layer. So, the hidden layer will contain 15 hidden nodes instead of ten hidden nodes that were specified in the third experiment.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as in the following table:

TABLE 8

Results of the fourth experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
90%	24	91.2%	22	73.2%	67	74.8%	63

The results of this experiment are some how better than the results of the third experiment. The training accuracy in both trials is very close and could be considered as high accuracy. On the other side, the testing accuracy is still low and there is a big gap between the training accuracy and the testing accuracy. It was obvious during running the code that the learning process is getting more slower, because of adding more nodes in the hidden nodes. So, the number of weights increases. It is also noticeable that the training accuracy in this experiment is lower than in the third experiment. So, increasing the number of nodes does not guarantee the correct convergence.

In this topology the number of weights is distributed as follows: 210 weights that connect the input layer with the hidden layer with additional 15 biased weights for the 15 hidden nodes and 75 weights that connect the hidden layer with the output layer with additional five biased weights for the five output nodes. The total number of weights is 305, so that the learning process takes more time and gets slower than what it takes in the third experiment.

Experiment 5:

In this experiment the network topology will be significantly changed, as a new hidden layer will be added. In this topology the first hidden layer will contain five nodes and the second hidden node will contain also five nodes.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as in the following table:

TABLE 9

Results of the fifth experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
89.2%	27	93.2%	17	76%	60	83.6%	41

It is noticeable that the results of this experiment are completely better than the results from the fourth experiment. The training accuracy in both trials is close to each other and the gap between the training accuracy and the testing accuracy is getting closer. But in general, the testing accuracy is still at a low rate. The second trial is considered the best result until now. It was obvious during running the code that the learning process is getting more slower, because of adding a new hidden layer to the topology. So, the number of weights increases. In this topology the number of weights is distributed as follows: 70 weights that connect the input layer with the first hidden layer with additional five biased weights for the five hidden nodes of the first hidden layer, 25 weights that connect the first hidden layer with the second hidden layer with additional five biased weights for the five hidden nodes of the second hidden layer and 25 weights that connect the second hidden layer with the output layer with additional five biased weights for the five output nodes. The total number of weights is 135.

Experiment 6:

In this experiment the network topology will be based on the topology created in the fifth experiment but the first hidden layer will contain ten hidden nodes. The second hidden layer will be the same with five hidden nodes.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as in the following table:

TABLE 10

Results of the sixth experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
94%	15	94.4%	14	80.4%	49	80.8%	48

The training accuracy is getting increased. But the most important observation from this experiment is the stability in the training accuracy and in the testing accuracy between the two trials. This could mean that the topology is approaching the convergence. The gap between the training accuracy and the testing accuracy is still high. The topology is now considered a complex one that takes more time to learn the weights. In this topology the number of weights is distributed as follows: 140 weights that connect the input layer with the first hidden layer with additional ten biased weights for the ten hidden nodes of the first hidden layer, 50 weights that connect the first hidden layer with the second hidden layer with additional five biased weights for the five hidden nodes of the second hidden layer and 25 weights that connect the second hidden layer with the output layer with additional five biased weights for the five output nodes. The total number of weights is 235.

Experiment 7:

In this experiment the network topology will be based on the topology created in the sixth experiment but the first hidden layer will contain 15 hidden nodes. The second hidden layer will be the same with five hidden nodes.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as in the following table:

TABLE 11

Results of the seventh experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
92.8%	18	87.2%	32	82%	45	72.8%	68

The results of this experiment are frustrating to some extent. After reaching some close accuracy rates between the training dataset and the testing dataset, the current result takes us backwards. The training accuracy and the testing accuracy are lower than in the previous experiment. But, there is one observation that could give us some hope, that is, the gap between the training accuracy and the testing accuracy is about 10% which is better than in the previous experiment. In this topology the number of weights is distributed as follows: 210 weights that connect the input layer with the first hidden layer with additional 15 biased weights for the 15 hidden nodes of the first hidden layer, 75 weights that connect the first hidden layer with the second hidden layer with additional five biased weights for the five hidden nodes of the second hidden layer and 25 weights that connect the second hidden layer with the output layer with additional five biased weights for the five output nodes. The total number of weights is 335.

Experiment 8:

In this experiment the network topology will be based on the topology created in the seventh experiment but the second hidden layer will contain ten hidden nodes. The first hidden layer will be the same with 15 hidden nodes.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as in the following table:

TABLE 12

Results of the eighth experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
99.6%	1	93.2%	17	98%	5	86%	35

The results of this experiment are completely encouraging. The training accuracy in the first trial reached to 99.6% which is the higher rate till now. There was only one data sample that was not classified correctly. On the other hand, the testing accuracy for the first trial was 98% which is considered the best result we got. In the second trial, the results were decreased. This proves again the fact that the initial weights have a big impact on the learning rate. In general this could be considered as admissible results, because the difference between the training accuracy and the testing accuracy is getting lower.

In this topology the number of weights is distributed as follows: 210 weights that connect the input layer with the first hidden layer with additional 15 biased weights for the 15 hidden nodes of the first hidden layer, 150 weights that connect the first hidden layer with the second hidden layer with additional ten biased weights for the ten hidden nodes of the second hidden layer and 50 weights that connect the second hidden layer with the output layer with additional five biased weights for the five output nodes. The total number of weights is 440. The complexity of the introduced topology in this experiment has affected the speed of the learning process, so it is now lower than before. It is now clear that more weights the network topology has, the lower the learning process is and the more accurate the convergence is.

Experiment 9:

In this experiment the network topology will be based on the topology created in the eighth experiment but the second hidden layer will contain 15 hidden nodes. The first hidden layer will be the same with 15 hidden nodes.

Results:

The code has been run twice for each dataset. The training and testing accuracy and the number of errors for each trial were computed as in the following table:

TABLE 13

Results of the ninth experiment for MLP

Training Accuracy				Testing Accuracy			
Trial A		Trial B		Trial A		Trial B	
Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors	Accuracy Rate	# of Errors
96.4%	9	99.6%	1	89.6%	26	98%	5

The results of this experiment are the acceptable results through our observations. The training accuracy is high in both trials. And the testing accuracy is also high in both trials. So, the best topology that has solved the cars classification problem consists of a multi-layer perceptron with two hidden layer and each hidden layer contains 15 hidden neurons. As this topology is the most complex one, the learning speed has reached the maximum slower rate.

In this topology the number of weights is distributed as follows: 210 weights that connect the input layer with the first hidden layer with additional 15 biased weights for the 15 hidden nodes of the first hidden layer, 225 weights that connect the first hidden layer with the second hidden layer with additional 15 biased weights for the ten hidden nodes.of the second hidden layer and 75 weights that connect the second hidden layer with the output layer with additional five biased weights for the five output nodes. The total number of weights is 545.

Final Observations:

The following charts show the training accuracy and the testing accuracy for the first and second trials. From the first chart, it is clear that the accuracy in the eighth experiment is the most optimum for the training and testing dataset (see figure 21). For the second trial, the ninth experiment has the higher accuracy (see figure 22). We should confirm again that the randomly initialized weights have affected the learning and testing accuracy, that's why the results are different between both trials. It is also expected that if a third trial has been implemented, the final results would also be different between the three trials.

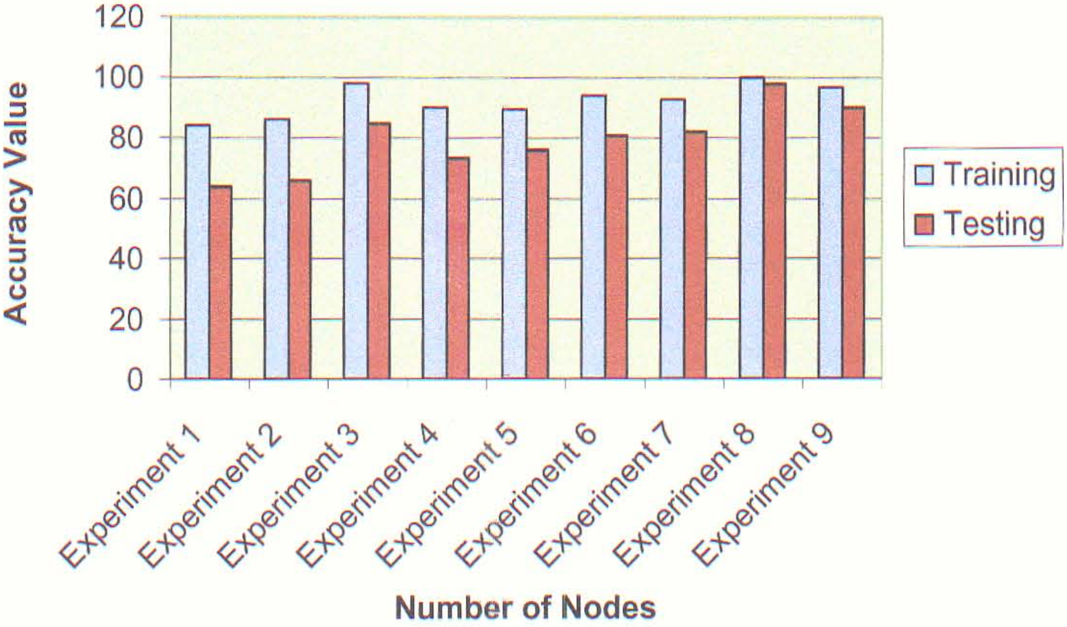


Figure 21. Flow Chart of the First Trial for the MLP

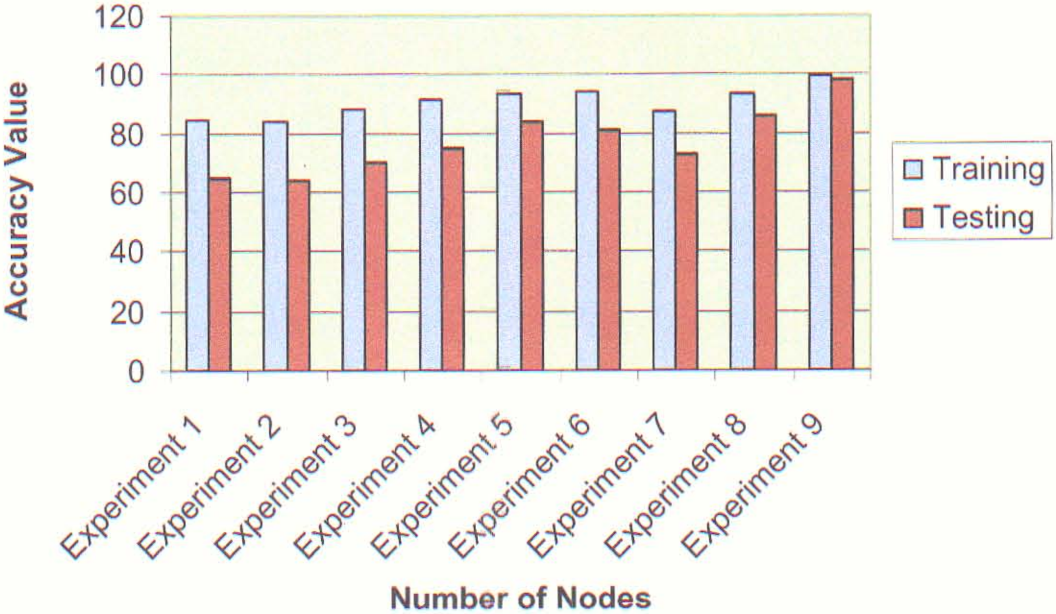


Figure 22. Flow Chart of the Second Trial for the MLP

All the experiments implemented here are following the trial and error method. This was done by using the code found in "Appendix H" that is implemented for each experiment separately.

Error Analysis:

As investigating the errors derived from the classification process is considered a very important phase in evaluating classification techniques, we introduce here the types of error that may be resulted from the experiments done in this thesis. In general, we have two error types, namely type-I errors and type-II errors. Type-I errors are the data samples that haven't been classified correctly with respect to the predefined class labels, where type-II errors are the data samples that shouldn't be classified correctly but have been classified correctly with respect to the predefined class labels. In any classification technique, type-1 errors are less dangerous than type-II errors. So if the result of a certain classification process results in many type-I errors but less type-II errors, then this is considered an advantage for the technique used and *vise versa.*

In the first trial of the ninth experiment, there were nine errors discovered in the training dataset, where there was only one error in the testing dataset. All these errors are type-I errors. We have constructed a confidence matrix for the errors derived from both trials of experiment 9. In figure 23 the confidence matrix for the training process of the first trial shows that all the samples of the first class have been classified correctly. In addition, 42 samples of the second class have been classified correctly and eight samples have not been classified correctly where four samples have been classified into the first class and another four samples have been classified into the third class. It is also obvious that 49 samples of the third class have been classified correctly where only one sample has not been classified correctly, as it has been classified into the first class. All the samples of the fourth and fifth classes have been classified correctly.

	1	2	3	4	5
1	50	0	0	0	0
2	4	42	4	0	0
3	1	0	49	0	0
4	0	0	0	50	0
5	0	0	0	0	50

Training-Trial A

	1	2	3	4	5
1	50	0	0	0	0
2	3	32	15	0	0
3	1	0	49	0	0
4	0	0	0	48	2
5	1	0	0	4	45

Testing-Trial A

Figure 23. Confidence Matrix of the Errors for the MLP Experiment 9 – Trial A

In figure 23 also, the confidence matrix of the testing process shows that all the samples of the first class have been classified correctly. 32 samples of the second class have been classified correctly and 18 samples have not been classified correctly where three samples have been classified into the first class and 15 samples have been classified into the third class. For the third class, 49 of its samples have been classified correctly and only one sample has not been classified correctly, as it has been classified into the first class. It is also clear that 48 samples of the fourth class have been classified correctly where only two samples have been classified into the fifth class. Finally, 45 samples of the fifth class have been classified correctly where four samples have been classified into the fourth class and only one sample has been classified into the first class.

The confidence matrices of the second trial can also be analyzed as the same way used in the first trial (see figure 24).

	1	2	3	4	5
1	50	0	0	0	0
2	0	50	4	0	0
3	1	0	49	0	0
4	0	0	0	50	0
5	0	0	0	0	50

Training-Trial B

	1	2	3	4	5
1	50	0	0	0	0
2	1	49	0	0	0
3	1	0	49	0	0
4	0	0	0	48	2
5	1	0	0	0	49

Testing-Trial B

Figure 24. Confidence Matrix of the Errors for the MLP Experiment 9 – Trial Testing the Scalability of the Multi-Layer Perceptrons Networks

As investigating the scalability and the efficiency of the techniques implemented in this thesis is considered the backbone of this thesis, we have implemented the most scalable and efficient algorithm here, namely the multi-layer perceptrons, using MatLab and a complex and large dataset. The MatLab code contains the same logic implemented in the previous experiment of the cars classification problems. The dataset contains two types of data, namely the training dataset which contains 6238 data sample and the testing dataset which contains 1559. Each dataset contains a huge number of features: 617 features. This dataset contains data about isolated spoken letters that need to be classified into 26 classes (one for each letter). All features are continuous, real-valued features scaled into the range -1.0 to 1.0. Some of these features are contour features, sonorant features, pre-sonorant features, and post-sonorant features, which are describing the difference between each spoken letter.

This data set was generated as follows: 150 subjects spoke the name of each letter of the alphabet twice. Hence, there are 52 training examples from each speaker. Speakers are grouped into sets of 30 speakers each, and are referred to as isolet1, isolet2, isolet3, isolet4, and isolet5. In our experiment we have considered the data collected from the first four groups to be the training dataset, and the data collected from the fifth group to be the testing dataset. The data appears in the training dataset in sequential order, first the speakers from isolet1, then isolet2, and so on.

The source of this dataset, namely the machine learning repository of the University of California, Irvine, has stated that it is a very good domain for noisy and perceptual task. In addition, it is a very good domain for testing the scaling abilities of classification algorithms. This is really the goal of this section in the thesis.

This dataset has been used by Fanty and Cole (1991), as they have targeted to predict which letter name was spoken. They have implemented the OPT backpropagation algorithm and the result was 95.9% of the data samples has been classified successfully. Their network architecture contained 56 hidden nodes and 26 output nodes (one per class). Dietterich and Bakiri have implemented the dataset in two different studies in 1991 and in 1994. In 1991, they used the OPT backpropagation algorithm with 78 hidden nodes and 26 output nodes. In this case the result was 95.83% correct. Then they have changed the network architecture to contain 156 hidden nodes and 30 output nodes and have used a 30-bit error-correcting output code with OPT. in this case the result was 96.73%. In 1994, Dietterich and Bakiri have used the dataset in a wide study, in which they have used different algorithms with many different configurations as follows:

TABLE 14

Results of Past Usage of the Isolated Spoken Letters Dataset

Algorithm and Configuration	Errors	Error Percentage	Correct Percentage
Opt 30-bit ECOC	51	3.27	96.73
Opt 62-bit ECOC	63	4.04	95.96
Opt OPC	65	4.17	95.83
C4.5 107-bit ECOC soft pruned	103	6.61	93.39
C4.5 92-bit ECOC soft pruned	107	6.86	93.14
C4.5 45-bit ECOC soft pruned	109	6.99	93.01
C4.5 107-bit ECOC soft raw	116	7.44	92.56
C4.5 92-bit ECOC soft raw	118	7.57	92.43
C4.5 30-bit ECOC soft raw	175	11.23	88.77
C4.5 30-bit ECOC hard pruned	185	11.87	88.13

It is very clear from the previous table that the implementations in which the backpropagation algorithm was used had the most higher correct classification rate, where in the case of using the C4.5 algorithm the error rate was getting increased. So, we can say that the backpropagation algorithm is more scalable and efficient than the C4.5 algorithm when classifying the data samples in the isolated spoken letters dataset.

In our implementation, the network architecture contained two hidden layers with 200 hidden nodes for each one. The output layer contained 26 neurons as we have 26 classes. After running the code, we got a very magnificent result: the training accuracy was 99.872% with only 8 error samples that were not classified correctly and the testing accuracy was 91.725% with only 129 error samples that were not classified correctly. These results have proved that the multi-layer perceptrons networks technique with backpropagation is the most scalable and efficient algorithm implemented in this thesis.

Geometric Representation of the Multi-Layer Perceptrons Networks

In order to understand the behind concepts of the MLP networks we need to introduce a geometric representation that helps in understanding the concepts of different layers in the topology. For delivering the right concept, we introduce here a certain classification problem which contains some data samples that need to be classified into three different classes. The given problem is a 2-D problem that could be presented in a scatter diagram. As the given problem is a simple one because the data samples are not mixed up, two classifiers are sufficient to solve the problem (see figure 25).

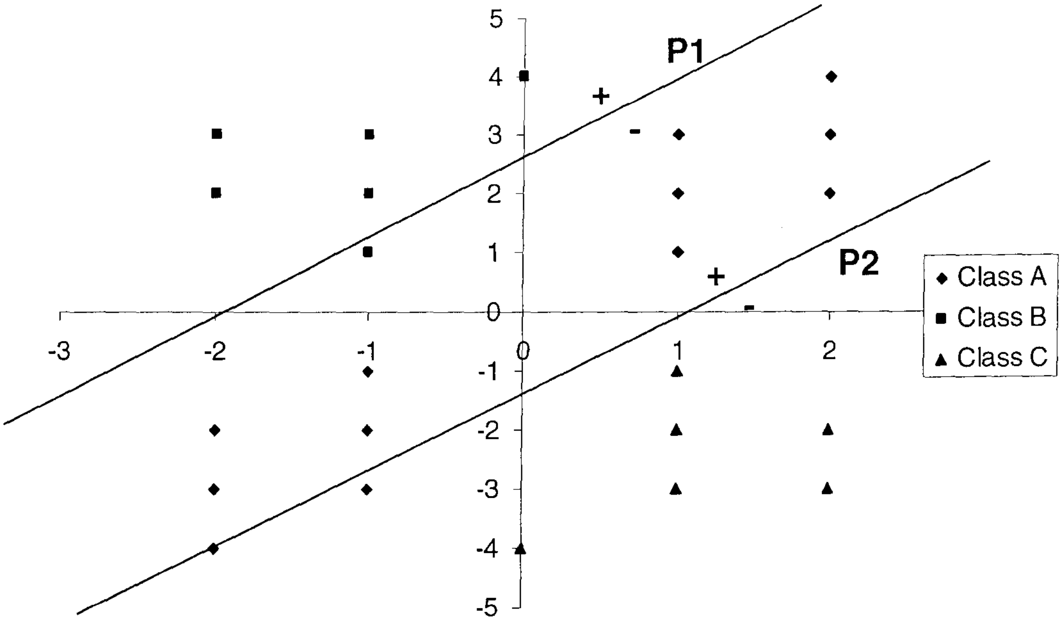


Figure 25. Geometric Representation of MLP Networks

Since two classifiers have been used to solve the problem, the first hidden layer which represents the partitioning layer will have two nodes (neurons). Each neuron represents a classifier.

In this problem we do not need to have a second hidden layer. This problem is a simple one that performs only an “anding process”. As the two classifiers divide the space into three regions, the output layer will have only three nodes. The output layer represents here the "anding layer". In some other complex problems it may represent the "oring layer" and the second hidden layer represents the "anding layer".

Now, we need to get the weights of both classifiers:

The classifier P1:

$y = mx + c$
$y = 1.3\ x + 2$
$y - 1.3\ x - 2 = 0$

$w_2x_2 + w_1x_1 + w_0 = 0$
$w_0 = -2$
$w_1 = -1.3$
$w_2 = 1$

The classifier P2:

$y = mx + c$
$y = 1.3\ x - 2$
$y - 1.3\ x + 2 = 0$

$w_2x_2 + w_1x_1 + w0 = 0$
$w_0 = 2$
$w_1 = -1.3$
$w_2 = 1$

The input layer could be modeled as in the following figure:

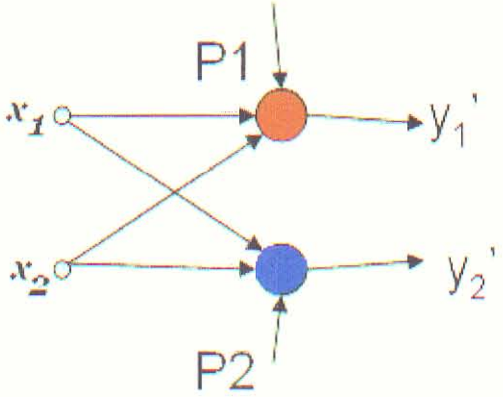


Figure 26. Geometric Representation of the Input Layer for MLP Networks

The output y_1', and y_2' is either +1 or -1 depending on the region where the input samples (x_1, and x_2) lies. There are four possible output combinations for y_1', and y_2' and each combination defines a region (class label) in the space. Note that the fourth combination does not define any class label because there is no region that could be defined by +1 for the output y_1' and -1 for the output y_2'. This could be checked in the following table.

TABLE 15

Results of Past Usage of the Isolated Spoken Letters Dataset

y_1'	y_2'	Region Code
-1	+1	R1 (class A)
+1	+1	R2 (class B)
-1	-1	R3 (class C)
+1	-1	-

y_1' and y_2' will be considered as inputs (x_1 and x_2) for the next layer, which is the output layer. This layer the “anding layer” which will have two inputs (y_1', y_2') and three outputs (we have only three classes) as specified in the following table:

TABLE 16

Truth Table of the Outputs of the Output Layer

y_1'	y_2'	Class Label	y_1	y_2	y_3
-1	+1	A	+1	-1	-1
+1	+1	B	-1	+1	-1
-1	-1	C	-1	-1	+1

Now the final topology will be as follows:

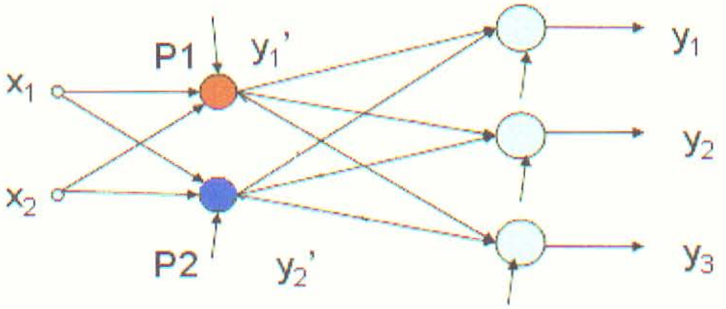


Figure 27. Geometric Representation of the Final Topology for MLP Networks

We need to get the weights of the output layer, so we need first to plot the points which formulate table 15 (see figure 28). Plotting the values of the weights helps us completely in understanding the concept behind this geometric representation. This scatter plot will also facilitate the plotting of the three perceptrons of the output layer as shown in the next coming figures.

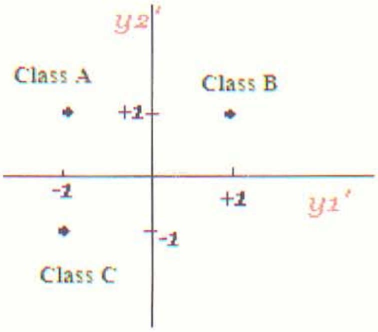


Figure 28. Scatter Plot of the Outputs of the Hidden Layer

To get the weights of the first neuron in the output Layer whose output is y_1, we need to get the weights of the perceptron that identifies y_1 (class A) as modeled in the following figure.

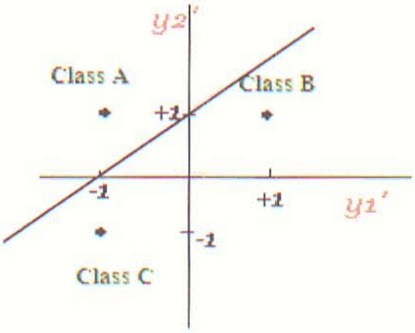


Figure 29. The Perceptron of the First Output Node

Its weights are calculated as follows:

$W_2 = 1, W_1 = -1, W_0 = -1$

To get the weights of the second neuron in the output Layer whose output is y2, we need to get the weights of the perceptron that identifies y_2 (class B) as modeled in the following figure.

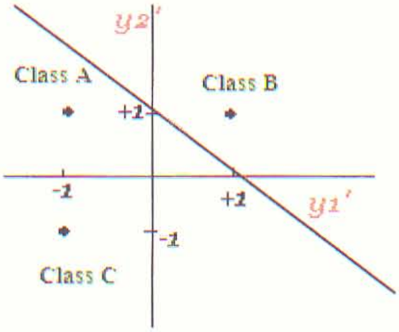


Figure 30. The Perceptron of the Second Output Node

Its weights are calculated as follows:

$W_2 = 1, W_1 = 1, W_0 = -1$

To get the weights of the third neuron in the output Layer whose output is y3, we need to get the weights of the perceptron that identifies y_3 (class C) as modeled in the following figure.

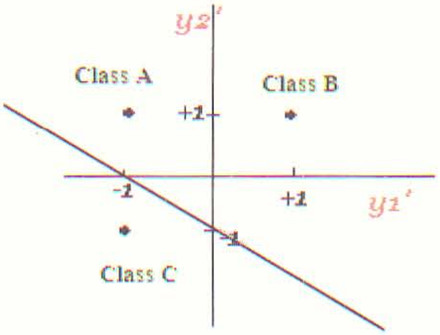


Figure 31. The Perceptron of the Third Output Node

Its weights are calculated as follows:

$W_2 = 1, W_1 = 1, W_0 = 1$

The benefits of the above analysis are:

1. Any MLP with two hidden layers, an input layer and an output layer could be designed to solve any classification problem. In case we have a more complex problem, we can use more nodes in the partitioning Layer, and perform the same analysis.
2. The number of nodes in the partitioning layer depends directly on the complexity of the problem.
3. The output layer (anding layer) is used to collect and gather contiguous and joint regions to map them to a particular class label.
4. In more complex problems the output layer represents the (oring layer) which is then used to gather the non-contiguous and disjoint regions to map them to a certain class label.

Performance Evaluation

Through the current chapter it becomes so clear how the multi-layer perceptron networks could solve most of the difficult and complex classification problems. In addition, we have seen through all the experiments which were implemented in this chapter how efficient the MLP with backpropagation algorithm is in finding weights for classification problems that are not linearly separable. As it is some how difficult to know if the given classification problem is linearly separable or not, it was discovered that the single-layer perceptron is not an efficient technique for classification. So that the MLP networks were introduced to help in solving those critical and mixed up problems. As it has been proved in the previous section of the concepts of the geometric representation of the MLP, a maximum of two hidden layers could be used in any neural network solution to solve any complex problem. If the problem is getting more complex, we can then add new nodes in the first hidden layer then in the second hidden layer until reaching the required convergence rate.

Because most of the MLP networks are dependent on the backpropagation algorithm to a high extent, the learning process in the MLP networks may be considered too slow for many applications and it scales up poorly as the classification problem becomes larger and more complex. The backpropagation learning algorithm runs faster than earlier learning methods such as the single-layer perceptron, but it is still much slower than what is desirable. Even in the case of having simple classification problems, the backpropagation algorithm requires the complete training dataset to be available many times through the learning process. This means that we are limited to testing small networks with only a few thousand trainable weights. As if the networks are getting larger, the performance of the learning process will be too slow. Some problems of the real world are in this simple and small scale, but most of the tasks for which neural networks technology might be appropriate are much too large and complex to be handled by our current backpropagation algorithm. Actually this could be solved through running the learning process on faster computers or to implement the network elements directly in VLSI chips which stand for very large scale integration chips. It is a technology that refers to semiconductor chips that are engineered to accommodate a large number of transistors and can do much more. But the best solution would be combining a powerful and faster hardware and a more reliable learning algorithm that scale up to very large networks in order to tackle a much larger environment of possible classification problems.

We have investigated how the MLP network has solved the XOR problem, but if our goal is to develop good learning algorithms for real-world pattern classification problems, the XOR problem is the wrong problem to concentrate on. Classification tasks take advantage of a learning network's ability to generalize from the input patterns it has seen during training to nearby patterns in the space of possible inputs. These kinds of networks must infrequently make precise distinctions between very similar input patterns, but this is the exception rather than the rule. The XOR problem, on the other hand, has exactly the opposite character: generalization is rarely implemented, since the nearest data samples of an input pattern belong to the opposite class label of the input pattern itself.

Although they have some disadvantages, the MLP networks with backpropagation algorithm are considered one of the most powerful and efficient techniques to classify data samples. So we can live with the slow process of learning in order to achieve the best accuracy desired in different applications.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

Final Conclusion

We have seen through this thesis chapter how the three algorithms are varied greatly in performance and in types of classification problems that could be solved. Many factors could be investigated in order to select the most suitable techniques that would help greatly in achieving the task. As we have started the analysis in this thesis from the less scalable algorithm, namely the nearest neighbor, to the most scalable and efficient one, namely the multi-layer perceptron, it was clearly obvious how the distribution of the given data samples plays a great role in deciding which algorithm to use. For instance, if the training samples are clearly separable, the nearest neighbor technique may be the most efficient one to classify the new unknown samples. If the given problem is linearly separable the task is to classify the data into two classes only, the single- layer perceptron may be the best technique to achieve the required classification task. On the other hand, if the problem we face is complicated with mixed up data samples that could not be linearly separable task, the multi-layer perceptrons networks may be the most suitable technique to solve the problem.

It is noticeable that in the previous paragraph we are using the words "may" and "could", this is because we can't guarantee that the choices here are the best effort done in this field. In classification domain, one can't guarantee any final results without testing or even without the trial-and-error technique. In practical, everything should be tested before starting the classification process. Of course, we do not mean to discourage people who are interested in this field, but we introduce here some facts about one of the most evolving fields in computer sciences. All the analysis achieved in this thesis will -with no doubt- help the researchers and interested people in the classification field in selecting the most efficient technique to address any classification tasks.

The nearest neighbor technique is a lazy learner as it needs to store all the training samples and does not build a classifier until a new unknown sample is available for classification. This technique is considered a slower classifier as it slows the speed of recognition, because it measures the distance between the unknown samples and all of the historically classified samples. Although the nearest neighbor technique is very slow, it is considered to be the most accurate classifier among the two other techniques introduced in this thesis. This is because it will result in at least one minimum distance between the new sample and the old samples.

The single-layer perceptron is more efficient that the nearest neighbor technique. It is one of the classification algorithms that learn the weights of the neural network designed for achieving the classification task. Once the learning process is finished, there is no further need for storing the training data samples as the case in the nearest neighbor method. This fact has resulted in having a quicker algorithm that completes the learning process quickly and efficiently. In order to have good results, the training data samples should be linearly separable. In this case the convergence rate of the algorithm is increased to a satisfactory value. But if the training data samples are mixed up and not linearly separable, the perceptron algorithm won't be useful in solving the problems, as it will not converge based on the desired rate. Most real life classification problems are not linearly separable, so that the single-layer perceptron is not the good choice for most of these problems. As a result, many studies and researches were continued in order to develop more scalable and efficient algorithms.

The last Algorithm represented in this thesis is the multi-layer perceptrons with backpropagation algorithm. This technique has overcome many deficiencies that were faced in implementing the two other techniques. The multi-layer perceptron networks have the power to classify the non-linearly separable data samples. So, the MLP networks were developed to solve those critical and very complex classification problems. This fact was derived from an advanced geometric representation of the MLP: a maximum of two hidden layers could be used in any neural network application to achieve any complex task for classification. But in the case of having more complex problems, we can add new neurons or nodes to the hidden layers. The experiments introduced in this thesis prove these facts clearly. They also prove that the trial-and-error technique is one of the best techniques to achieve the desired results. The convergence rates achieved by the multi-layer perceptrons are more efficient and with some little efforts and trial, we can reach the rate we need. In general, the MLP networks are considered to be slower learner as it scales up poorly when the given classification problem becomes larger and more complex. Although the backpropagation algorithm is faster than the other learning algorithms introduced in this thesis, it is still much slower than what is desirable. Of course, the studies in this field are continued to find more efficient methods for classification.

Future Work

The development of new learning algorithms is being a critical task nowadays. Many classification algorithms are being used in current studies. Some have gained big success and others have failed to achieve what is desirable. A more consistent way to develop a new approach in the classification field is to try to update some existing algorithms to reflect all the changes needed. So, the adaptive neural networks are considered a focal point for our future work. This new approach has been tackled in some previous studies but it still need some efforts to get a more efficient and scalable architecture. The adaptive architecture will be of a great benefit for the classification systems of medical analysis and GIS. This could be the start point of our future work. Another target would be investigating and analyzing the effect of the learning rate in the multi-layer perceptrons architecture, as determining the best learning rate would help extremely in improving the performance of the network.

REFERENCES

Abdel Azim, H. (2004). Handouts and Lectures of Neural Networks at the Regional Information Technology Institute, Cairo, Egypt.

Battiti, R. (1989). Accelerated Backpropagation Learning: Two Optimization Methods. Complex Systems 3, 331-342.

Brank, J., Grobelnik, M., Milić-Frayling, N., & Mladenić, D. (2002). Feature Selection Using Linear Support Vector Machines. Technical Report MSR-TR-2002-63. Redmond, WA: Microsoft Research. ftp://ftp.research.microsoft.com/pub/tr/tr-2002-63.pdf.

Center for Vision Speech & Signal Processing (1996). Pattern Recognition and Neural Networks. Surrey, UK: University of Surrey. http://www.ee.surrey.ac.uk/Research/VSSP/report96/report/node19.html.

Chen, Y.Q. (1995). Novel Techniques for Image Texture Classification. PhD Thesis, Department of Electronics and Computer Science, University of Southampton. http://citeseer.ist.psu.edu/chen95novel.html.

Cover, T.M., & Hart, P.E. (1967). Nearest Neighbor Pattern Classification. IEEE Transactions on Information Theory IT-13(1), 21-27. http://yreka.stanford.edu/~cover/papers/transIT/0021cove.pdf.

Dietterich, T. G., & Bakiri, G. (1991) Error-Correcting Output Codes: A General Method for Improving Multiclass Inductive Learning Programs. Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91), Anaheim, CA: AAAI Press.

Dietterich, T. G., & Bakiri, G. (1994) Solving Multiclass Learning Problems via Error-Correcting Output Codes. ftp://ftp.cs.orst.edu/pub/tgd/papers/tr-ecoc.ps.gz

Domeniconi, C., Peng, J., & Gunopulos, D. (2002). Locally Adaptive Metric Nearest-Neighbor Classification. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(9), 1281-1285. http://www.ise.gmu.edu/~carlotta/publications/pami.pdf.

Eaton, H., & Olivier, T. (1992). Learning Coefficient Dependence on Training Set Size, Neural Networks 5, 283-288.

Fahlman, S.E. (1988). An Empirical Study of Learning Speeds in Backpropagation Networks. Technical Report CMU-Cs-88-162. Pittsburgh, PA: Carnegie Mellon University.

Fanty, M., & Cole, R. (1991). Spoken letter recognition. In Lippman, R. P., Moody, J., & Touretzky, D. S. (Eds). Advances in Neural Information Processing Systems 3. San Mateo, CA: Morgan Kaufmann.

Freund, Y., & Schapire, R.E. (1998). Large Margin Classification Using the Perceptron Algorithm. Computational Learning Theory, 209-217. http://citeseer.ist.psu.edu/freund98large.html.

Fukumizu, K. (1996). Active Learning in Multilayer Perceptrons. Advances in Neural Information Processing Systems 8, 295-301. Cambridge, MA: MIT Press. http://citeseer.ist.psu.edu/fukumizu96active.html.

Guyon, I., & Elisseeff, A. (2003). An Introduction to Variable and Feature Selection. Journal of Machine Learning Research 3, 1157-1182. http://jmlr.csail.mit.edu/papers/volume3/guyon03a/guyon03a.pdf.

Han, E., Karypis, G., & Kumar, V. (1999). Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification. Technical Report TR 99-019. Minneapolis, MN: Department of Computer Science and Engineering, University of Minnesota. https://wwws.cs.umn.edu/tech_reports_upload/tr1999/99-019.pdf.

Han, J., & Kamber, M. (2001). Data Mining: Concepts and Techniques. San Francisco, CA: Morgan Kaufmann Publishers.

Hettich, S., & Blake, C.L., & Merz, C.J. (1998). UCI Repository of Machine Learning Databases. Irvine, CA: University of California, Department of Information and Computer Science. http://www.ics.uci.edu/~mlearn/MLRepository.html

Huang, S., hung, Y. (1991). Bounds on the Number of Hidden Neurons in Multilayer Perceptrons. IEEE Transactions on Neural Networks, 2(1), 47-55.

Jacobs, R.A. (1988). Increased Rates of Convergence Through Learning Rate Adaptation. Neural Networks 1, pp. 295–307.

Jain, A.K., Mao, J., & Mohiuddin, K.M. (1996). Artificial Neural Networks: A Tutorial. IEEE Computer 29(3), 31-44. http://citeseer.ist.psu.edu/jain96artificial.html.

Mashor, M.Y. (2000). Performance Comparison Between Back Propagation, PRE and MRPE Algorithms for Training MLP Networks. International Journal of the Computer, the Internet and Management (IJCIM) 8(3). http://www.journal.au.edu/ijcim/2000/sep00/mohd_yusoff.pdf.

McAuley, D., Dennis, S. (1997, updated in 1999). The Backpropagation Network: Learning by Example. Connectionist Models of Cognition. http://www.itee.uq.edu.au/~cogs2010/cmc/chapters/BackProp.

Michie, D., Spiegelhalter, D., & Taylor, C. (Eds). (1994). Machine Learning, Neural, and Statistical Classification. New York: Ellis Horwood. http://www.amsta.leeds.ac.uk/~charles/statlog.

Minsky, M., & Papert, S. (1969). Perceptrons: An Introduction to Computational Geometry. Cambridge, MA: MIT Press.

Mirchandani, G., & Cao, W. (1989). On Hidden Nodes for Neural Networks. IEEE Transactions on Circuits and Systems 36(5), 661-664.

Moody, J. (1994). Prediction Risk and Architecture Selection for Neural Networks. From Statistics to Neural Networks: Theory and Pattern Recognition Applications, 147-165. Berlin: Springer-Verlag, NATO ASI Series F. http://citeseer.ist.psu.edu/moody94prediction.html.

Murray, A.F., & Edwards, P.J. (1993). Synaptic Weight Noise During MLP Learning Enhances Fault-Tolerance, Generalisation and Learning Trajectory. Advances in Neural Information Processing Systems 5, 491-498. San Meteo, CA: Morgan Kaufmann. http://citeseer.ist.psu.edu/murray93synaptic.html.

Plagianakos, V.P., Magoulas, G.D., & Varhatis, M.N. (2001). Learning Rate Adaption in Stochastic Gradient Descent. Advances in convex analysis and global optimization 54, 433-444. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Pollack, J.B. Book Review: Marvin L. Minsky and Seymour A. Papert. Perceptrons: An Introduction to Computational Geometry. http://citeseer.ist.psu.edu/29184.html.

Reisinger, J., Stanley, L.O., & Miikkulainen, R. (2004). Evolving Reusable Neural Models. Proceedings of the Genetic and Evolutionary Computation Conference (GECCo-2004). New York, NY: Springer-Verlag. http://nn.cs.utexas.edu/downloads/papers/reisinger.gecco04.pdf

Riedmiller, M. (1994). Advanced Supervised Learning in Multi-layer Perceptrons - From Backpropagation to Adaptive Learning Algorithms. International Journal of Computer Standards and Interfaces 16, 265-278. http://citeseer.ist.psu.edu/riedmiller94advanced.html.

Rummelhart, D.E., McClelland, J.L., & the PDP Group (Ed.) (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge, MA: MIT Press.

Sarle, W.S. (1994). Neural Networks and Statistical Models. Proceedings of the Nineteenth Annual SAS Users Group International Conference, 1538--1550. Cary, NC: SAS Institute. http://citeseer.ist.psu.edu/sarle94neural.html.

Schiffmann, W., Joost, M., & Werner, R. (1992). Synthesis and Performance Analysis of Multilayer Neural Network Architecture. Technical report 16/1992. http://citeseer.ist.psu.edu/78140.html.

Sheng, M.A., & Chuanyi, J.I. (1999). Performance and Efficiency: Recent Advances in Supervised Learning. Proceedings of the IEEE 87(9), 1519-1535. http://users.ece.gatech.edu/~jic/procieee99.pdf.

Singh, S., Haddon, J., & Markou, M. (1999). Nearest Neighbor Strategies for Image Understanding. Proceedings of Workshop on Advanced Concepts for Intelligent Vision, Systems (ACIVS'99). Baden-Baden. http://www.dcs.ex.ac.uk/research/pann/pdf/pann_SS_004.pdf.

Salzberg, S.L. (1997). Methodological Note On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach. Data Mining and Knowledge Discovery 1, 317-328. Boston, MA: Kluwer Academic Publishers. http://www.cogsci.ucsd.edu/~rik/courses/cogs200-w05/readings/salzberg97-compClassif.pdf

Vogl, T.P., Mangis, J.K., Rigler, J.K., Zink, W.t., & Alkon, D.L (1988). Accelerating the Convergence of the Back–propagation Method, Biological Cybernetics 59, 257–263.

Wang, J., & Gasser, L. (2002). Mutual Online Concept Learning for Multiple Agents. Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems - Bologna, Italy -, 362-369. New York, NY: ACM Press. http://www.isrl.uiuc.edu/~gasser/papers/wang02aamas.pdf.

Weiss, S.M., & Kulikowski, C.A. (1991). Computer systems that learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. San Mateo, CA: Morgan Kaufmann.

APPENDICES

Appendix A

Dataset of the Nearest Neighbor Practical Experiment

Weight	Height	Class
35.25	135.97	1
36.35	142.53	1
43.36	149.54	1
40.33	147.79	1
43.22	135.81	1
49.64	130.06	1
38.2	145.9	1
30.04	139.71	1
35.5	143.17	1
31.71	136.09	1
43.29	130.01	1
43.34	135.22	1
36.88	133.62	1
30.7	142.08	1
42.7	132.15	1
39.35	130.02	1
49.36	149.09	1
33.9	143.18	1
37.45	140.61	1
34.34	132.83	1
47.47	139.67	1
45.96	138.64	1
38.54	130.64	1
43.61	136.76	1
35.3	130.37	1
40.1	148.39	1
37.3	140.88	1
32.68	149.21	1
59.07	193.23	2
47.37	202.2	2
48.04	186.55	2
56.41	193.26	2
55.53	186.06	2
41.99	187.28	2
53.15	186.06	2
49.13	185.06	2

57.35	201.02	2
53.09	189.14	2
51.61	194.54	2
43.97	195.77	2
59.8	191.32	2
42.87	193.27	2
53	195.8	2
50.97	187.14	2
46.29	196.51	2
45.43	190.74	2
58.31	196.31	2
47.96	202.55	2
49.85	199.49	2
43.99	189.58	2
41.37	190.26	2
55.86	198.99	2
45.73	203.67	2
47.88	197.35	2
49.44	201.88	2
42.4	195.11	2
86.94	175.46	3
91.6	196.04	3
89.45	178.55	3
95.25	189.37	3
95.05	199.16	3
92.48	188.51	3
92.09	191.11	3
102.43	186.74	3
92.07	198.83	3
97.18	199.33	3
104.95	188.5	3
93.42	184.43	3
102.98	190.57	3
97.14	185.59	3
94.75	189.83	3
96.08	181.77	3
88.65	177.74	3
95.52	197.38	3
85.37	191.49	3
95.32	193.05	3
94.86	197	3
85.95	176.46	3
102.16	196.8	3

103.71	188.74	3
104.96	182.69	3
92.47	178.95	3
95.52	198.41	3
103.83	182.51	3
99.04	152.61	4
103.59	158.09	4
99.11	142.85	4
108.52	160	4
114.04	157.3	4
107.37	146.87	4
98.29	157.49	4
109.28	147.91	4
100.46	140.44	4
114.98	153.15	4
111.6	142.43	4
112.36	150.21	4
97.44	151.3	4
112.63	152.91	4
110.74	153.16	4
111.12	156.15	4
114.62	145.76	4
110.01	158.22	4
98.41	140.17	4
112.32	153.13	4
112.91	152.06	4
111.31	142.46	4
99.97	159.65	4
101.53	154.57	4
110.69	143.89	4
109.68	146.64	4
95.19	145.15	4
98.27	149.07	4

Appendix B

MATLAB Code for the Nearest Neighbor Classifiers

```
% This code illustrates the Nearest Neighbor Classification Technique

% Class 1: Jockey
% Class 2: Basket Ball
% Class 3: Rugby
% Class 4: Judo


function [minDistance, classLabel] = Nearest_Neighbor_Classifier(dataset,unknown)

  count = 1;
  newValue = 0;
  minDistance = 0;
  dataSize = size(dataset);

  for i = 1 : length(dataset)

      for j = 1 : dataSize(2)

          if j < dataSize(2)

            % I check the value of j, in case that the computed
            % weight is 0. In this case it should compute the height
            % and not to compute again the weight.

            if newValue == 0 & j == 1

                newValue = abs(unknown(1) - dataset(i,j));

            else

                newValue = newValue + abs(unknown(2) - dataset(i,j));

            end

        end

    end
```

```
        % Getting the lowest value and the class label

        if minDistance == 0
              minDistance = newValue;
              classLabel = dataset(i,j);
        end

        if minDistance < newValue
              minDistance = minDistance;
              classLabel = classLabel;
        else
              minDistance = newValue;
              classLabel = dataset(i,j);
        end

        newValue = 0;
        count = count + 1;
end
```

MATLAB Code 1 – Nearest Neighbor Classifier

In order to run the code in the MATLAB environment we need to follow the following steps:

- Copy the code
- Open MATLAB and create a new m file in the same folder that contains the dataset file "SPlayers.txt"
- Paste the code in the new created m file
- Save the m file and name it "Nearest_Neighbor_Classifier.m"
- In the workspace of MATLAB type the following:

  ```
  dataFile = load('SPlayers.txt');

  unknownSample = [70 175];

  [minDistance classLabel] = Nearest_Neighbor_Classifier (dataFile,
  unknownSample)
  ```

- Change the values of the unknown sample and run the code again

Appendix C

Case Study for the Perceptron Learning Algorithm

Initial weights:

$y = mx + c$
$y = 1.5x + 0$
$y - 1.5x = 0$

$f(\underline{x}) = f(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_0$
$w_0 = 0$
$w_1 = - 1.5$
$w_2 = 1$
$\beta = 0.2$

First Iteration:

Sample 1:

$Ya = g(f) = g(w_0 + w_1 x_1 + w_2 x_2)$
$Ya = g(0 + (-1.5) \text{ x } 1 + (1 \text{ x } 2))$
$Ya = g(0.5) = +1$
$Yd = +1$
$\delta = Yd - Ya = 0$

Sample 2:

$Ya = g(f) = g(w_0 + w_1 x_1 + w_2 x_2)$
$Ya = g(0 + (-1.5) \text{ x } (-1) + (1 \text{ x } 2))$
$Ya = g(3.5) = +1$
$Yd = -1$
$\delta = Yd - Ya = (-1) - (+1) = -2$
$\Delta w_1 = \beta\, \delta\, x_1 = 0.2\, (-2) -1 = 0.4$
$\Delta w_2 = \beta\, \delta\, x_2 = 0.2\, (-2)\, 2 = - 0.8$
$\Delta w_0 = \beta\, \delta = 0.2\, (-2) = - 0.4$

$(w_1)_{new} = (w_1)_{old} + \Delta w_1 = - 1.5 + 0.4 = - 1.1$
$(w_2)_{new} = (w_2)_{old} + \Delta w_2 = 1 + (-0.8) = 0.2$
$(w_0)_{new} = (w_0)_{old} + \Delta w_0 = 0 + (-0.4) = - 0.4$

Sample 3:

$Ya = g(f) = g(w_0 + w_1x_1 + w_2x_2)$

$Ya = g((-0.4) + (-1.1) \text{ x } (0) + (0.2 \text{ x } -1))$

$Ya = g(-0.6) = -1$

$Yd = -1$

$\delta = Yd - Ya = 0$

Second Iteration:

Sample 1:

$Ya = g(f) = g(w_0 + w_1x_1 + w_2x_2)$

$Ya = g((-0.4) + (-1.1) \text{ x } (1) + (0.2 \text{ x } 2))$

$Ya = g(-1.1) = -1$

$Yd = +1$

$\delta = Yd - Ya = (+1) - (-1) = 2$

$\Delta w_1 = \beta \delta x_1 = 0.2\ (2)\ 1 = 0.4$

$\Delta w_2 = \beta \delta x_2 = 0.2\ (2)\ 2 = 0.8$

$\Delta w_0 = \beta \delta = 0.2\ (2) = 0.4$

$(w_1)_{new} = (w_1)_{old} + \Delta w_1 = -1.1 + 0.4 = -0.7$

$(w_2)_{new} = (w_2)_{old} + \Delta w_2 = 0.2 + 0.8 = 1$

$(w_0)_{new} = (w_0)_{old} + \Delta w_0 = -0.4 + (0.4) = 0$

Sample 2:

$Ya = g(f) = g(w_0 + w_1x_1 + w_2x_2)$

$Ya = g(0 + (-0.7) \text{ x } (-1) + (1 \text{ x } 2))$

$Ya = g(2.7) = +1$

$Yd = -1$

$\delta = Yd - Ya = (-1) - (+1) = -2$

$\Delta w_1 = \beta \delta x_1 = 0.2\ (-2)\ -1 = 0.4$

$\Delta w_2 = \beta \delta x_2 = 0.2\ (-2)\ 2 = -0.8$

$\Delta w_0 = \beta \delta = 0.2\ (-2) = -0.4$

$(w_1)_{new} = (w_1)_{old} + \Delta w_1 = -0.7 + 0.4 = 1.1$

$(w_2)_{new} = (w_2)_{old} + \Delta w_2 = 1 + (-0.8) = 0.2$

$(w_0)_{new} = (w_0)_{old} + \Delta w_0 = 0 + (-0.4) = -0.4$

Sample 3:

$Ya = g(f) = g(w_0 + w_1x_1 + w_2x_2)$

$Ya = g((-0.4) + (1.1) \text{ x } (0) + (0.2 \text{ x } -1))$

$Ya = g(-0.6) = -1$

$Yd = -1$

$\delta = Yd - Ya = 0$

Appendix D

Dataset of the Practical Experiment for Perceptron Learning Algorithm

Weight	Height	Class
35.25	135.97	-1
36.35	142.53	-1
43.36	149.54	-1
40.33	147.79	-1
43.22	135.81	-1
49.64	130.06	-1
38.20	145.9	-1
30.04	139.71	-1
35.50	143.17	-1
31.71	136.09	-1
43.29	130.01	-1
43.34	135.22	-1
36.88	133.62	-1
30.70	142.08	-1
42.70	132.15	-1
39.35	130.02	-1
49.36	149.09	-1
33.90	143.18	-1
37.45	140.61	-1
34.34	132.83	-1
47.47	139.67	-1
45.96	138.64	-1
38.54	130.64	-1
43.61	136.76	-1
35.30	130.37	-1
40.10	148.39	-1
37.30	140.88	-1
32.68	149.21	-1
86.94	175.46	1
91.60	196.04	1

89.45	178.55	1
95.25	189.37	1
95.05	199.16	1
92.48	188.51	1
92.09	191.11	1
102.43	186.74	1
92.07	198.83	1
97.18	199.33	1
104.95	188.5	1
93.42	184.43	1
102.98	190.57	1
97.14	185.59	1
94.75	189.83	1
96.08	181.77	1
88.65	177.74	1
95.52	197.38	1
85.37	191.49	1
95.32	193.05	1
94.86	197	1
85.95	176.46	1
102.16	196.8	1
103.71	188.74	1
104.96	182.69	1
92.47	178.95	1
95.52	198.41	1
103.83	182.51	1

Appendix E

MATLAB Code of the Practical Experiment for Perceptron Learning Algorithm

```
%This code illustrates the Perceptron Learning Algorithm

% Class 1: Jockey, Desired Output is -1
% Class 2: Rugby, Desired Output is +1

load SportsPlayers_Perceptron.txt;  % Loading the dataset
[n m] = size(SportsPlayers_Perceptron);
Features = SportsPlayers_Perceptron (:,1:2);
Vlabels = SportsPlayers_Perceptron (:,3);
j = 0;

for i = 1:n
   if Vlabels(i) == 1
      j = j+1;
      Rugby(j,:) = Features(i,:);
   else
      Jockeys(i,:) = Features(i,:);
   end;
end;

itermax=500;

% Initializing the weights and the learning rate (beta) randomly
wo = rand;
w1 = rand;
w2 = rand;
SumDelta = 1;
iter = 0;
beta = 0.3;

% Plotting the data samples on a scatter diagram
axis([0 200 0 250]);
hold on;

X = Rugby(:,1);
Y = Rugby(:,2);
plot(X,Y,'rx:');

X = Jockeys(:,1);
Y = Jockeys(:,2);
plot(X,Y,'bo:');
```

```
% Starting the Perceptron Algorithm
while SumDelta ~= 0 & iter < itermax
   iter =iter + 1;
   SumDelta=0;

   for i=1:n
      x1 = Features(i,1);
      x2 = Features(i,2);
      f = wo + w1 * x1 + w2 * x2;
      yactual = hardlims(f);
      ydesired = Vlabels(i);
      delta = ydesired - yactual;
      SumDelta = abs(SumDelta) + abs(delta);

      if (delta ~= 0)
         dwo = beta * delta *1;
         dw1 = beta * delta * x1;
         dw2 = beta * delta * x2;

         wo = wo + dwo;
         w1 = w1 + dw1;
         w2 = w2 + dw2;
     end;
   end;
end;

%Plotting the perceptron
        cx2 = -wo / w2;% Part cut from x2 axis
        cx1 = -wo / w1;% Part cut from x1 axis
        cx1up = -(wo + w2 * 250) / w1;% Cutting Upper axis
        cx2rt = -(wo + w1 * 200) / w2;% Cutting Upper axis
        line([cx1 cx1up],[0 250]);
```

In order to run the code in the MATLAB environment we need to follow the following steps:

- Copy the code
- Open MATLAB and create a new m file in the same folder that contains the dataset file “SportsPlayers_Perceptron.txt”
- Paste the code in the new created m file

- Save the m file and name it "Sports_Players_Perceptron.m"
- Run the code using F5 button

Appendix F

Case Study for the Multi-Layer Perceptron

Calculating the input into the first hidden node (Z_1) in the hidden layer. This input represents the output of the first input node in the input layer:

$$Y_{H1} = g(f) = g(w_0 + w_1x_1 + w_2x_2)$$
$$Y_{H1} = g(0 + (1 \text{ x } 1) + (-1 \text{ x } 0.5))$$
$$Y_{H1} = g(1 - 0.5) = 0.5$$
$$Y_{H1} = g(0.5) = 1/1 + e^{-0.5} = 0.622$$

Calculating the input into the second hidden node (Z_2) in the hidden layer. This input represents the output of the second input node in the input layer:

$$Y_{H2} = g(f) = g(w_0 + w_1x_1 + w_2x_2)$$
$$Y_{H2} = g(-1.5 + (0.5 \text{ x } 1) + (0.5 \text{ x } -0.5))$$
$$Y_{H2} = g(-1.5 + 0.5 - 0.25) = -1.25$$
$$Y_{H2} = g(-1.25) = 1/1 + e^{1.25} = 0.223$$

Calculating the actual output for the first output node in the output layer:

$$Y_{a1} = g(f) = g(w_0 + w_1x_1 + w_2x_2)$$
$$Y_{a1} = g(0 + (2 \text{ x } 0.622) + (1 \text{ x } 0.223))$$
$$Y_{a1} = g(1.244 + 0.223) = 1.467$$
$$Y_{a1} = g(1.467) = 1/1 + e^{-1.467} = 0.813$$
$$Y_1 = 0.8$$

Calculating the error at the first output node:

$$\delta_1 = Y_1 - Y_{a1} = 0.8 - 0.813 = - 0.013$$

Calculating the actual output for the second output node in the output layer:

$$Y_{a2} = g(f) = g(w_0 + w_1x_1 + w_2x_2)$$
$$Y_{a2} = g(-0.5 + (1.5 \text{ x } 0.622) + (-1 \text{ x } 0.223))$$
$$Y_{a2} = g(-0.5 + 0.933 - 0.223) = 0.21$$

$Y_{a2} = g(0.21) = 1/1 + e^{-0.21} = 0.552$
$Y_2 = 0.2$

Calculating the error at the second output node:

$\delta_2 = Y_2 - Y_{a2} = 0.2 - 0.552 = - 0.352$

Applying the delta rule to compute the new weights for the first output node:

$\Delta w_i = \beta\delta(Y_a(1-Y_a))x_i$

$\Delta\ W_{11} = 0.3 \text{ x } - 0.013 \text{ x } (0.813 \text{ x } (1 - 0.813)) \text{ x } 0.622$
$\Delta\ W_{11} = - 0.000369$

$(W_{11})_{new} = W_{11} + \Delta\ W_{11}$
$(W_{11})_{new} = 2 - 0.000369$
$(W_{11})_{new} = 1.99$

$\Delta\ W_{21} = 0.3 \text{ x } - 0.013 \text{ x } (0.813 \text{ x } (1 - 0.813)) \text{ x } 0.223$
$\Delta\ W_{21} = - 0.00582$

$(W_{21})_{new} = W_{21} + \Delta\ W_{21}$
$(W_{21})_{new} = 1 - 0.00582$
$(W_{21})_{new} = 0.000132$

Applying the delta rule to compute the new weights for the second output node:

$\Delta w_i = \beta\delta(Y_a(1-Y_a))x_i$

$\Delta\ W_{12} = 0.3 \text{ x } - 0.352 \text{ x } (0.552 \text{ x } (1 - 0.552)) \text{ x } 0.622$
$\Delta\ W_{12} = - 0.0162$

$(W_{12})_{new} = W_{12} + \Delta\ W_{12}$
$(W_{12})_{new} = 1.5 - 0.0162$
$(W_{12})_{new} = 1.48$

$\Delta\ W_{22} = 0.3 \text{ x } - 0.352 \text{ x } (0.552 \text{ x } (1 - 0.552)) \text{ x } 0.223$
$\Delta\ W_{22} = - 0.00582$

$(W_{22})_{new} = W_{22} + \Delta\ W_{22}$
$(W_{22})_{new} = -1 - 0.00582$
$(W_{22})_{new} = - 1.00582$

Now I compute the backpropagated error at the first hidden node:

$\delta_3 = ((W_{11})_{new} \text{ x } \delta_1) + ((W_{12})_{new} \text{ x } \delta_2)$

$\delta_3 = (1.99 \text{ x } - 0.013) + (1.48 \text{ x } - 0.352)$

$\delta_3 = - 0.02587 + (- 0.52096)$

$\delta_3 = - 0.547$

Now I compute the backpropagated error at the second hidden node:

$\delta_4 = ((W_{21})_{new} \text{ x } \delta_1) + ((W_{22})_{new} \text{ x } \delta_2)$

$\delta_4 = (0.000132 \text{ x } - 0.013) + (- 1.00582 \text{ x } - 0.352)$

$\delta_4 = - 0.0000017 + 0.354$

$\delta_4 = - 0.354$

Applying the delta rule to compute the new weights for the first hidden node:

$$\Delta w_i = \beta\delta(Y_{H1}(1 - Y_{H1}))x_i$$

$\Delta\ V_{11} = 0.3 \text{ x } - 0.547 \text{ x } (0.622 \text{ x } (1 - 0.622)) \text{ x } 1$

$\Delta\ V_{11} = - 0.0386$

$(V_{11})_{new} = V_{11} + \Delta\ V_{11}$

$(V_{11})_{new} = 1 - 0.0386$

$(V_{11})_{new} = 0.961$

$\Delta\ V_{21} = 0.3 \text{ x } - 0.547 \text{ x } (0.622 \text{ x } (1 - 0.622)) \text{ x } 0.5$

$\Delta\ V_{21} = - 0.0193$

$(V_{21})_{new} = V_{21} + \Delta\ V_{21}$

$(V_{21})_{new} = - 1 - 0.0193$

$(V_{21})_{new} = - 1.019$

Applying the delta rule to compute the new weights for the second hidden node:

$$\Delta w_i = \beta\delta(Y_{H2}(1 - Y_{H2}))x_i$$

$\Delta\ V_{12} = 0.3 \text{ x } - 0.354 \text{ x } (0.223 \text{ x } (1 - 0.223)) \text{ x } 1$

$\Delta\ V_{12} = - 0.0184$

$(V_{12})_{new} = V_{12} + \Delta\ V_{12}$

$(V_{12})_{new} = 0.5 - 0.0184$

$(V_{12})_{new} = 0.482$

$\Delta\ V_{22} = 0.3 \text{ x } - 0.354 \text{ x } (0.\ 223 \text{ x } (1 - 0.223)) \text{ x } 0.5$
$\Delta\ V_{22} = -\ 0.0092$

$(V_{22})_{new} = V_{22} + \Delta\ V_{22}$
$(V_{22})_{new} = -\ 0.5 - 0.0092$
$(V_{22})_{new} = -\ 0.509$

Appendix G

Training Dataset of the Practical Experiment for the Multi-Layer Perceptron

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.3916	0.69985	0.63514	0.40391	0.33929	0.24264	0.66714	0.75103	0.45737	0.56548	0.52958	0.54312	0.54312	0.22706	1
0.11571	0.20679	0.18767	0.11934	0.10025	0.071693	0.19712	0.22191	0.13514	0.16708	0.15647	0.16048	0.16048	0.06709	1
0.20729	0.37047	0.33621	0.21381	0.1796	0.12844	0.35315	0.39756	0.24211	0.29934	0.28033	0.2875	0.2875	0.1202	1
-0.026552	-0.047452	-0.043065	-0.027387	-0.023005	-0.016452	-0.045234	-0.050922	-0.031011	-0.038341	-0.035907	-0.036825	-0.036825	-0.015396	1
0.67399	1.2045	1.0931	0.69518	0.58395	0.41761	1.1482	1.2926	0.78719	0.97325	0.91146	0.93476	0.93476	0.3908	1
-0.59179	-1.0576	-0.95983	-0.6104	-0.51273	-0.36668	-1.0082	-1.135	-0.69119	-0.85456	-0.8003	-0.82076	-0.82076	-0.34314	1
-0.058724	-0.10495	-0.095244	-0.06057	-0.050879	-0.036386	-0.10004	-0.11262	-0.068587	-0.084798	-0.079414	-0.081445	-0.081445	-0.03405	1
0.0091106	0.016282	0.014777	0.0093971	0.0078936	0.0056451	0.015521	0.017473	0.010641	0.013156	0.012321	0.012636	0.012636	0.0052826	1
-0.062844	-0.11231	-0.10193	-0.06482	-0.054449	-0.038939	-0.10706	-0.12053	-0.0734	-0.090749	-0.084987	-0.08716	-0.08716	-0.036439	1
-0.25457	-0.45496	-0.41289	-0.26258	-0.22057	-0.15774	-0.4337	-0.48823	-0.29733	-0.36761	-0.34427	-0.35307	-0.35307	-0.14761	1
0.040502	0.072384	0.065691	0.041776	0.035092	0.025096	0.069	0.077677	0.047305	0.058486	0.054773	0.056173	0.056173	0.023484	1
0.66969	1.1968	1.0862	0.69074	0.58023	0.41495	1.1409	1.2844	0.78217	0.96704	0.90564	0.9288	0.9288	0.3883	1
-0.0259	-0.046288	-0.042008	-0.026715	-0.02244	-0.016048	-0.044124	-0.049672	-0.03025	-0.0374	-0.035026	-0.035921	-0.035921	-0.015018	1
0.28588	0.51091	0.46367	0.29487	0.24769	0.17713	0.48703	0.54827	0.33389	0.41282	0.3866	0.39649	0.39649	0.16576	1
0.67082	1.1989	1.088	0.69192	0.58121	0.41565	1.1428	1.2865	0.7835	0.96869	0.90718	0.93038	0.93038	0.38897	1
0.64831	1.1586	1.0515	0.6687	0.5617	0.4017	1.1045	1.2434	0.7572	0.93617	0.87674	0.89915	0.89915	0.37591	1
0.66679	1.1917	1.0815	0.68776	0.57772	0.41315	1.136	1.2788	0.77878	0.96286	0.90173	0.92478	0.92478	0.38662	1
0.62647	1.1196	1.0161	0.64618	0.54279	0.38817	1.0673	1.2015	0.7317	0.90465	0.84721	0.86887	0.86887	0.36325	1
0.50722	0.90649	0.82267	0.52317	0.43947	0.31428	0.86412	0.97278	0.59242	0.73244	0.68594	0.70348	0.70348	0.2941	1
0.47448	0.84798	0.76957	0.48941	0.4111	0.294	0.80834	0.90999	0.55418	0.68517	0.64167	0.65807	0.65807	0.27512	1
0.57888	1.0346	0.9389	0.59709	0.50155	0.35868	0.9862	1.1102	0.67611	0.83592	0.78285	0.80286	0.80286	0.33565	1
0.49645	0.88723	0.8052	0.51206	0.43013	0.30761	0.84576	0.95211	0.57983	0.71689	0.67137	0.68853	0.68853	0.28786	1
0.57109	1.0206	0.92625	0.58904	0.4948	0.35385	0.97292	1.0953	0.66701	0.82466	0.7723	0.79205	0.79205	0.33113	1
-0.49992	-0.89344	-0.81083	-0.51564	-0.43314	-0.30976	-0.85168	-0.95877	-0.58389	-0.7219	-0.67606	-0.69335	-0.69335	-0.28987	1
0.079448	0.14199	0.12886	0.081947	0.068835	0.049227	0.13535	0.15237	0.092793	0.11473	0.10744	0.11019	0.11019	0.046067	1

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
-0.014161	-0.025308	-0.022968	-0.014606	-0.012269	-0.0087743	-0.024125	-0.027158	-0.016539	-0.020449	-0.01915	-0.01964	-0.01964	-0.008211	1
0.2609	0.46627	0.42316	0.26911	0.22605	0.16166	0.44448	0.50037	0.30472	0.37675	0.35283	0.36185	0.36185	0.15128	1
0.6353	1.1354	1.0304	0.65527	0.55043	0.39364	1.0823	1.2184	0.742	0.91738	0.85914	0.8811	0.8811	0.36837	1
0.65519	1.1709	1.0627	0.67579	0.56767	0.40596	1.1162	1.2566	0.76524	0.94611	0.88604	0.90869	0.90869	0.3799	1
0.6743	1.2051	1.0937	0.69551	0.58423	0.41781	1.1488	1.2932	0.78756	0.97371	0.91189	0.9352	0.9352	0.39098	1
0.56522	1.0101	0.91673	0.58299	0.48971	0.35022	0.96292	1.084	0.66015	0.81619	0.76436	0.78391	0.78391	0.32773	1
0.68421	1.2228	1.1097	0.70573	0.59281	0.42395	1.1656	1.3122	0.79914	0.98802	0.92529	0.94895	0.94895	0.39673	1
0.68988	1.2329	1.1189	0.71157	0.59772	0.42746	1.1753	1.3231	0.80575	0.9962	0.93295	0.95681	0.95681	0.40001	1
0.62289	1.1132	1.0103	0.64248	0.53968	0.38595	1.0612	1.1946	0.72751	0.89947	0.84236	0.8639	0.8639	0.36117	1
-0.24417	-0.43637	-0.39602	-0.25185	-0.21155	-0.15129	-0.41597	-0.46828	-0.28518	-0.35258	-0.3302	-0.33864	-0.33864	-0.14158	1
0.39849	0.71217	0.64632	0.41102	0.34526	0.24691	0.67888	0.76424	0.46542	0.57543	0.5389	0.55267	0.55267	0.23106	1
0.46877	0.83776	0.7603	0.48351	0.40615	0.29045	0.7986	0.89902	0.5475	0.67691	0.63393	0.65014	0.65014	0.27181	1
0.10835	0.19364	0.17573	0.11176	0.093876	0.067135	0.18459	0.2078	0.12655	0.15646	0.14653	0.15027	0.15027	0.062825	1
0.64011	1.144	1.0382	0.66024	0.5546	0.39662	1.0905	1.2276	0.74762	0.92434	0.86565	0.88778	0.88778	0.37116	1
-0.011147	-0.019921	-0.018079	-0.011497	-0.0096578	-0.0069067	-0.01899	-0.021378	-0.013019	-0.016096	-0.015074	-0.01546	-0.01546	-0.0064633	1
0.69098	1.2349	1.1207	0.71271	0.59867	0.42814	1.1772	1.3252	0.80704	0.99779	0.93444	0.95833	0.95833	0.40065	1
0.0053047	0.0094803	0.0086037	0.0054715	0.0045961	0.0032869	0.0090372	0.010174	0.0061957	0.0076601	0.0071737	0.0073572	0.0073572	0.0030758	1
0.64419	1.1513	1.0448	0.66444	0.55813	0.39915	1.0975	1.2355	0.75239	0.93022	0.87116	0.89343	0.89343	0.37352	1
0.24867	0.44441	0.40332	0.25649	0.21545	0.15408	0.42364	0.47691	0.29043	0.35908	0.33628	0.34488	0.34488	0.14419	1
0.58564	1.0466	0.94985	0.60405	0.5074	0.36287	0.9977	1.1232	0.684	0.84567	0.79198	0.81223	0.81223	0.33957	1
0.2194	0.39209	0.35584	0.22629	0.19009	0.13594	0.37377	0.42077	0.25625	0.31681	0.2967	0.30428	0.30428	0.12721	1
0.67033	1.198	1.0872	0.69141	0.58078	0.41534	1.142	1.2856	0.78292	0.96797	0.90651	0.92969	0.92969	0.38868	1
0.58154	1.0393	0.9432	0.59982	0.50385	0.36033	0.99072	1.1153	0.67921	0.83975	0.78644	0.80654	0.80654	0.33719	1
0.50451	0.90164	0.81827	0.52037	0.43711	0.3126	0.85949	0.96757	0.58925	0.72852	0.68227	0.69971	0.69971	0.29253	1
0.39913	0.71331	0.64735	0.41168	0.34581	0.24731	0.67997	0.76547	0.46617	0.57635	0.53976	0.55356	0.55356	0.23143	1
0.92642	0.55912	0.56834	0.67227	0.66928	0.39676	0.64625	0.6733	0.69204	0.62745	0.64737	0.67944	0.62922	0.3591	2
1.4117	0.85201	0.86608	1.0244	1.0199	0.60461	0.9848	1.026	1.0546	0.95615	0.9865	1.0354	0.95885	0.54721	2
1.3577	0.81941	0.83294	0.98524	0.98087	0.58148	0.94711	0.98675	1.0142	0.91956	0.94875	0.99576	0.92216	0.52627	2
1.4395	0.86876	0.8831	1.0446	1.0399	0.61649	1.0042	1.0462	1.0753	0.97494	1.0059	1.0557	0.9777	0.55797	2
0.66498	0.40133	0.40795	0.48255	0.48041	0.28479	0.46388	0.48329	0.49674	0.45038	0.46468	0.4877	0.45165	0.25776	2
0.80417	0.48534	0.49335	0.58356	0.58097	0.34441	0.56098	0.58445	0.60072	0.54466	0.56195	0.58979	0.54619	0.31171	2

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.25575	0.15435	0.1569	0.18559	0.18476	0.10953	0.17841	0.18587	0.19105	0.17322	0.17872	0.18757	0.17371	0.099133	2
1.07	0.64579	0.65645	0.77649	0.77304	0.45827	0.74644	0.77768	0.79933	0.72473	0.74773	0.78478	0.72677	0.41477	2
1.2494	0.75402	0.76647	0.90662	0.90259	0.53507	0.87154	0.90801	0.93329	0.84618	0.87304	0.9163	0.84857	0.48428	2
1.4546	0.87791	0.8924	1.0556	1.0509	0.62299	1.0147	1.0572	1.0866	0.98521	1.0165	1.0668	0.988	0.56384	2
1.2418	0.74946	0.76183	0.90114	0.89713	0.53184	0.86626	0.90252	0.92764	0.84106	0.86776	0.91076	0.84344	0.48135	2
0.80618	0.48655	0.49458	0.58501	0.58241	0.34527	0.56237	0.58591	0.60222	0.54601	0.56335	0.59126	0.54756	0.31249	2
1.2016	0.72521	0.73718	0.87198	0.86811	0.51463	0.83824	0.87332	0.89763	0.81385	0.83969	0.88129	0.81615	0.46577	2
-0.53302	-0.32169	-0.327	-0.38679	-0.38507	-0.22828	-0.37182	-0.38739	-0.39817	-0.36101	-0.37247	-0.39092	-0.36203	-0.20661	2
1.2201	0.73639	0.74854	0.88542	0.88148	0.52256	0.85115	0.88677	0.91146	0.82639	0.85263	0.89487	0.82873	0.47295	2
1.459	0.88056	0.89509	1.0588	1.0541	0.62487	1.0178	1.0604	1.0899	0.98818	1.0196	1.0701	0.99097	0.56554	2
1.1399	0.68793	0.69928	0.82715	0.82348	0.48817	0.79514	0.82842	0.85148	0.77201	0.79652	0.83598	0.77419	0.44183	2
1.0416	0.62862	0.63899	0.75584	0.75248	0.44608	0.72658	0.75699	0.77807	0.70545	0.72784	0.7639	0.70744	0.40373	2
-0.23134	-0.13962	-0.14193	-0.16788	-0.16713	-0.099079	-0.16138	-0.16813	-0.17282	-0.15669	-0.16166	-0.16967	-0.15713	-0.089673	2
1.2676	0.76505	0.77767	0.91988	0.91579	0.5429	0.88428	0.92129	0.94693	0.85855	0.88581	0.92969	0.86098	0.49136	2
0.57697	0.34821	0.35396	0.41868	0.41682	0.2471	0.40248	0.41932	0.431	0.39077	0.40318	0.42315	0.39187	0.22364	2
0.29665	0.17903	0.18199	0.21526	0.21431	0.12705	0.20693	0.21559	0.2216	0.20091	0.20729	0.21756	0.20148	0.11498	2
1.3165	0.79455	0.80766	0.95535	0.9511	0.56383	0.91837	0.95681	0.98345	0.89166	0.91996	0.96554	0.89418	0.5103	2
-0.53897	-0.32528	-0.33065	-0.39111	-0.38937	-0.23083	-0.37597	-0.39171	-0.40261	-0.36503	-0.37662	-0.39528	-0.36606	-0.20891	2
0.67932	0.40998	0.41675	0.49296	0.49077	0.29094	0.47388	0.49371	0.50746	0.46009	0.4747	0.49822	0.46139	0.26332	2
1.2995	0.78425	0.79719	0.94297	0.93878	0.55652	0.90647	0.94441	0.9707	0.8801	0.90804	0.95303	0.88259	0.50369	2
1.4457	0.87253	0.88693	1.0491	1.0445	0.61917	1.0085	1.0507	1.08	0.97917	1.0103	1.0603	0.98194	0.56039	2
0.64399	0.38866	0.39508	0.46732	0.46524	0.2758	0.44923	0.46804	0.48107	0.43617	0.45001	0.47231	0.4374	0.24962	2
1.3048	0.78748	0.80048	0.94685	0.94264	0.55882	0.91021	0.9483	0.9747	0.88373	0.91178	0.95696	0.88623	0.50577	2
1.4673	0.88556	0.90018	1.0648	1.06	0.62842	1.0236	1.0664	1.0961	0.9938	1.0253	1.0761	0.9966	0.56876	2
1.3219	0.79781	0.81098	0.95927	0.95501	0.56615	0.92214	0.96074	0.98748	0.89532	0.92374	0.96951	0.89785	0.5124	2
-0.70911	-0.42796	-0.43503	-0.51457	-0.51229	-0.30369	-0.49466	-0.51536	-0.52971	-0.48027	-0.49551	-0.52006	-0.48163	-0.27486	2
0.95103	0.57397	0.58344	0.69013	0.68706	0.4073	0.66342	0.69119	0.71043	0.64412	0.66457	0.6975	0.64594	0.36864	2
0.14922	0.090058	0.091544	0.10828	0.1078	0.063907	0.10409	0.10845	0.11147	0.10106	0.10427	0.10944	0.10135	0.05784	2
0.62116	0.37489	0.38107	0.45076	0.44875	0.26603	0.43331	0.45145	0.46401	0.42071	0.43406	0.45557	0.42189	0.24077	2
0.54881	0.33122	0.33669	0.39825	0.39648	0.23504	0.38284	0.39886	0.40996	0.3717	0.3835	0.4025	0.37275	0.21273	2
1.0717	0.6468	0.65747	0.7777	0.77424	0.45898	0.7476	0.77889	0.80057	0.72585	0.74889	0.786	0.7279	0.41541	2

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
-0.5423	-0.32729	-0.33269	-0.39353	-0.39178	-0.23225	-0.3783	-0.39413	-0.4051	-0.36729	-0.37895	-0.39773	-0.36833	-0.2102	2
0.90892	0.54855	0.5576	0.65957	0.65664	0.38927	0.63404	0.66058	0.67897	0.6156	0.63514	0.66661	0.61733	0.35231	2
-0.32577	-0.19661	-0.19986	-0.2364	-0.23535	-0.13952	-0.22725	-0.23676	-0.24335	-0.22064	-0.22765	-0.23892	-0.22126	-0.12627	2
0.61092	0.36871	0.37479	0.44333	0.44136	0.26164	0.42617	0.444	0.45636	0.41377	0.42691	0.44806	0.41494	0.2368	2
1.1722	0.70743	0.7191	0.8506	0.84682	0.50201	0.81768	0.8519	0.87561	0.79389	0.81909	0.85967	0.79613	0.45435	2
0.52567	0.31725	0.32249	0.38146	0.37976	0.22513	0.3667	0.38204	0.39268	0.35603	0.36733	0.38553	0.35703	0.20376	2
1.4759	0.89075	0.90546	1.071	1.0663	0.6321	1.0296	1.0727	1.1025	0.99962	1.0314	1.0825	1.0024	0.57209	2
1.0669	0.64389	0.65451	0.7742	0.77076	0.45692	0.74423	0.77538	0.79697	0.72258	0.74552	0.78246	0.72463	0.41354	2
-0.095333	-0.057536	-0.058485	-0.06918	-0.068872	-0.040829	-0.066502	-0.069286	-0.071214	-0.064568	-0.066618	-0.069918	-0.06475	-0.036953	2
1.4474	0.87354	0.88796	1.0503	1.0457	0.61989	1.0097	1.0519	1.0812	0.9803	1.0114	1.0615	0.98307	0.56104	2
-1.0898	-0.65771	-0.66857	-0.79082	-0.7873	-0.46673	-0.76021	-0.79203	-0.81408	-0.7381	-0.76153	-0.79926	-0.74018	-0.42242	2
1.393	0.8407	0.85458	1.0108	1.0063	0.59658	0.97172	1.0124	1.0406	0.94345	0.9734	1.0216	0.94612	0.53995	2
1.1602	0.70018	0.71174	0.84188	0.83814	0.49687	0.8093	0.84317	0.86665	0.78576	0.8107	0.85087	0.78798	0.4497	2
0.28993	0.22649	0.2155	0.2451	0.21351	0.17772	0.23561	0.23969	0.23689	0.22876	0.21786	0.21971	0.21648	0.16717	3
0.27092	0.21164	0.20137	0.22903	0.19951	0.16606	0.22016	0.22398	0.22136	0.21376	0.20358	0.20531	0.20229	0.15621	3
0.69039	0.53933	0.51315	0.58363	0.50841	0.42318	0.56104	0.57076	0.56408	0.54472	0.51878	0.52318	0.51549	0.39807	3
1.2668	0.98961	0.94157	1.0709	0.93287	0.77649	1.0295	1.0473	1.035	0.99951	0.95191	0.95998	0.94586	0.73041	3
1.1875	0.92768	0.88264	1.0039	0.87449	0.7279	0.96503	0.98174	0.97026	0.93696	0.89234	0.8999	0.88667	0.6847	3
1.0166	0.79414	0.75559	0.85938	0.74861	0.62312	0.82612	0.84042	0.83059	0.80209	0.76389	0.77037	0.75904	0.58614	3
0.74379	0.58105	0.55283	0.62877	0.54773	0.45591	0.60444	0.6149	0.60771	0.58686	0.55891	0.56365	0.55536	0.42886	3
-0.30803	-0.24063	-0.22894	-0.26039	-0.22683	-0.18881	-0.25032	-0.25465	-0.25167	-0.24303	-0.23146	-0.23342	-0.22999	-0.1776	3
-0.40283	-0.31469	-0.29941	-0.34054	-0.29665	-0.24692	-0.32736	-0.33303	-0.32913	-0.31784	-0.3027	-0.30527	-0.30078	-0.23227	3
1.0839	0.84672	0.80561	0.91627	0.79817	0.66437	0.88081	0.89606	0.88558	0.85518	0.81446	0.82136	0.80928	0.62494	3
0.44653	0.34883	0.33189	0.37748	0.32883	0.2737	0.36287	0.36915	0.36484	0.35231	0.33554	0.33838	0.3334	0.25746	3
-0.0006014	-0.0004698	-0.000447	-0.0005084	-0.0004429	-0.0003686	-0.0004887	-0.0004972	-0.0004914	-0.0004745	-0.0004519	-0.0004557	-0.000449	-0.0003468	3
0.20594	0.16088	0.15307	0.1741	0.15166	0.12623	0.16736	0.17026	0.16826	0.16249	0.15475	0.15606	0.15377	0.11874	3
1.1615	0.90739	0.86333	0.98192	0.85537	0.71198	0.94392	0.96026	0.94903	0.91646	0.87282	0.88022	0.86727	0.66972	3
1.2443	0.97201	0.92482	1.0519	0.91628	0.76268	1.0111	1.0287	1.0166	0.98173	0.93498	0.94291	0.92904	0.71742	3
0.76794	0.59991	0.57078	0.64918	0.56551	0.47071	0.62406	0.63486	0.62744	0.6059	0.57705	0.58194	0.57338	0.44278	3
1.2191	0.95236	0.90613	1.0306	0.89776	0.74727	0.99071	1.0079	0.99607	0.96189	0.91608	0.92385	0.91026	0.70292	3
1.2314	0.96193	0.91523	1.0409	0.90678	0.75477	1.0007	1.018	1.0061	0.97155	0.92529	0.93313	0.91941	0.70998	3

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.34897	0.27261	0.25938	0.29501	0.25698	0.2139	0.28359	0.2885	0.28512	0.27534	0.26223	0.26445	0.26056	0.20121	3
0.24786	0.19363	0.18423	0.20953	0.18253	0.15193	0.20142	0.20491	0.20251	0.19556	0.18625	0.18783	0.18507	0.14291	3
0.78363	0.61216	0.58244	0.66245	0.57706	0.48033	0.63681	0.64783	0.64026	0.61828	0.58884	0.59383	0.5851	0.45182	3
0.65768	0.51377	0.48883	0.55598	0.48432	0.40313	0.53446	0.54371	0.53735	0.51891	0.4942	0.49839	0.49106	0.3792	3
1.2566	0.98168	0.93402	1.0623	0.92539	0.77027	1.0212	1.0389	1.0267	0.99149	0.94428	0.95228	0.93828	0.72455	3
-0.06806	-0.053168	-0.050587	-0.057536	-0.05012	-0.041718	-0.055309	-0.056266	-0.055608	-0.0537	-0.051143	-0.051576	-0.050818	-0.039242	3
0.70968	0.5544	0.52748	0.59994	0.52261	0.435	0.57672	0.5867	0.57984	0.55994	0.53328	0.5378	0.52989	0.40919	3
1.0536	0.82305	0.78309	0.89066	0.77587	0.6458	0.85619	0.87102	0.86083	0.83129	0.7917	0.79841	0.78667	0.60748	3
1.1773	0.91968	0.87503	0.99523	0.86696	0.72162	0.95671	0.97328	0.96189	0.92888	0.88465	0.89215	0.87903	0.6788	3
1.2674	0.99006	0.942	1.0714	0.9333	0.77685	1.0299	1.0478	1.0355	0.99997	0.95235	0.96042	0.94629	0.73074	3
1.2319	0.96232	0.91559	1.0414	0.90714	0.75507	1.0011	1.0184	1.0065	0.97194	0.92566	0.9335	0.91977	0.71026	3
0.17043	0.13314	0.12668	0.14408	0.12551	0.10447	0.1385	0.1409	0.13925	0.13447	0.12807	0.12915	0.12725	0.098267	3
0.7844	0.61277	0.58302	0.6631	0.57764	0.4808	0.63744	0.64847	0.64089	0.6189	0.58942	0.59442	0.58568	0.45227	3
1.0823	0.84552	0.80447	0.91497	0.79704	0.66343	0.87956	0.89479	0.88433	0.85398	0.81331	0.8202	0.80814	0.62406	3
1.2254	0.95725	0.91078	1.0359	0.90237	0.7511	0.99579	1.013	1.0012	0.96682	0.92078	0.92859	0.91493	0.70652	3
0.31104	0.24298	0.23118	0.26294	0.22905	0.19065	0.25276	0.25714	0.25413	0.24541	0.23372	0.23571	0.23224	0.17934	3
0.75815	0.59226	0.56351	0.64091	0.55831	0.46472	0.61611	0.62678	0.61945	0.59819	0.5697	0.57453	0.56608	0.43714	3
0.56208	0.43909	0.41777	0.47516	0.41392	0.34453	0.45677	0.46468	0.45924	0.44348	0.42236	0.42594	0.41968	0.32408	3
-0.27003	-0.21095	-0.2007	-0.22827	-0.19885	-0.16552	-0.21944	-0.22324	-0.22063	-0.21306	-0.20291	-0.20463	-0.20162	-0.15569	3
-0.31326	-0.24472	-0.23283	-0.26482	-0.23069	-0.19201	-0.25457	-0.25898	-0.25595	-0.24716	-0.23539	-0.23739	-0.2339	-0.18062	3
-0.1999	-0.15616	-0.14858	-0.16899	-0.14721	-0.12253	-0.16245	-0.16526	-0.16333	-0.15773	-0.15021	-0.15149	-0.14926	-0.11526	3
0.5912	0.46184	0.43942	0.49977	0.43536	0.36238	0.48043	0.48875	0.48303	0.46646	0.44424	0.44801	0.44142	0.34087	3
0.56302	0.43983	0.41848	0.47596	0.41461	0.34511	0.45754	0.46546	0.46001	0.44423	0.42307	0.42666	0.42038	0.32463	3
0.96907	0.75703	0.72027	0.81921	0.71362	0.594	0.78751	0.80114	0.79177	0.7646	0.72819	0.73436	0.72356	0.55874	3
0.61942	0.48388	0.46039	0.52363	0.45614	0.37968	0.50336	0.51208	0.50609	0.48872	0.46545	0.46939	0.46249	0.35714	3
1.1411	0.89143	0.84815	0.96465	0.84032	0.69945	0.92732	0.94337	0.93234	0.90034	0.85747	0.86474	0.85202	0.65794	3
0.94143	0.73544	0.69973	0.79585	0.69327	0.57706	0.76505	0.77829	0.76919	0.74279	0.70742	0.71342	0.70292	0.54281	3
0.66918	0.52276	0.49738	0.5657	0.49278	0.41018	0.5438	0.55322	0.54675	0.52798	0.50284	0.5071	0.49965	0.38583	3
0.2738	0.21389	0.2035	0.23146	0.20162	0.16783	0.2225	0.22635	0.2237	0.21603	0.20574	0.20748	0.20443	0.15787	3
0.9268	0.72401	0.68886	0.78348	0.6825	0.56809	0.75316	0.7662	0.75724	0.73125	0.69643	0.70233	0.692	0.53438	3
0.45083	0.35219	0.33509	0.38112	0.33199	0.27634	0.36637	0.37271	0.36835	0.35571	0.33877	0.34164	0.33662	0.25994	3

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.94872	0.74114	0.70515	0.80201	0.69864	0.58153	0.77098	0.78432	0.77515	0.74855	0.7129	0.71894	0.70837	0.54702	3
0.43075	0.44724	0.48238	0.50945	0.60228	0.8161	0.4274	0.34307	0.52443	0.57058	0.50316	0.49698	0.49698	0.94658	4
0.83709	0.86913	0.93741	0.99002	1.1704	1.5859	0.83058	0.66669	1.0191	1.1088	0.9778	0.96578	0.96578	1.8395	4
0.10438	0.10837	0.11689	0.12345	0.14594	0.19775	0.10357	0.08313	0.12708	0.13826	0.12192	0.12042	0.12042	0.22937	4
-0.13505	-0.14022	-0.15123	-0.15972	-0.18882	-0.25586	-0.134	-0.10756	-0.16442	-0.17888	-0.15775	-0.15581	-0.15581	-0.29677	4
0.90451	0.93913	1.0129	1.0698	1.2647	1.7137	0.89747	0.72038	1.1012	1.1981	1.0565	1.0436	1.0436	1.9877	4
0.33427	0.34706	0.37433	0.39533	0.46737	0.6333	0.33167	0.26622	0.40696	0.44277	0.39045	0.38566	0.38566	0.73455	4
0.44996	0.46718	0.50388	0.53216	0.62913	0.85248	0.44646	0.35836	0.54781	0.59602	0.52559	0.51913	0.51913	0.98878	4
0.21693	0.22523	0.24293	0.25656	0.30331	0.411	0.21524	0.17277	0.26411	0.28735	0.2534	0.25028	0.25028	0.47671	4
0.62916	0.65324	0.70456	0.7441	0.8797	1.192	0.62427	0.50108	0.76599	0.8334	0.73492	0.72589	0.72589	1.3826	4
0.72154	0.74916	0.80802	0.85336	1.0089	1.367	0.71593	0.57466	0.87846	0.95577	0.84283	0.83247	0.83247	1.5856	4
-0.34581	-0.35905	-0.38726	-0.40899	-0.48352	-0.65517	-0.34312	-0.27542	-0.42102	-0.45807	-0.40394	-0.39898	-0.39898	-0.75993	4
0.29137	0.30252	0.32628	0.3446	0.40739	0.55202	0.2891	0.23205	0.35473	0.38595	0.34034	0.33616	0.33616	0.64028	4
0.51938	0.53926	0.58163	0.61427	0.7262	0.98401	0.51534	0.41365	0.63234	0.68798	0.60668	0.59923	0.59923	1.1413	4
0.55412	0.57532	0.62053	0.65535	0.77477	1.0498	0.54981	0.44132	0.67463	0.73399	0.64726	0.63931	0.63931	1.2177	4
0.71459	0.74194	0.80023	0.84514	0.99914	1.3539	0.70903	0.56912	0.87	0.94656	0.83471	0.82445	0.82445	1.5703	4
0.89827	0.93264	1.0059	1.0624	1.256	1.7018	0.89128	0.71541	1.0936	1.1899	1.0493	1.0364	1.0364	1.9739	4
0.377	0.39143	0.42218	0.44587	0.52712	0.71426	0.37407	0.30025	0.45899	0.49938	0.44037	0.43496	0.43496	0.82846	4
0.79339	0.82375	0.88847	0.93834	1.1093	1.5031	0.78722	0.63188	0.96594	1.0509	0.92675	0.91536	0.91536	1.7435	4
-0.62252	-0.64635	-0.69713	-0.73625	-0.87041	-1.1794	-0.61768	-0.4958	-0.75791	-0.8246	-0.72716	-0.71823	-0.71823	-1.368	4
0.86697	0.90015	0.97087	1.0254	1.2122	1.6425	0.86023	0.69048	1.0555	1.1484	1.0127	1.0003	1.0003	1.9052	4
-0.20917	-0.21718	-0.23424	-0.24738	-0.29246	-0.39629	-0.20754	-0.16659	-0.25466	-0.27707	-0.24433	-0.24133	-0.24133	-0.45965	4
0.22956	0.23835	0.25708	0.2715	0.32098	0.43493	0.22778	0.18283	0.27949	0.30408	0.26815	0.26486	0.26486	0.50447	4
0.027921	0.028989	0.031267	0.033022	0.039039	0.052898	0.027704	0.022237	0.033993	0.036984	0.032614	0.032213	0.032213	0.061356	4
0.46789	0.4858	0.52396	0.55337	0.65421	0.88646	0.46425	0.37264	0.56965	0.61978	0.54654	0.53982	0.53982	1.0282	4
-0.21418	-0.22238	-0.23985	-0.25331	-0.29947	-0.40579	-0.21252	-0.17058	-0.26076	-0.28371	-0.25018	-0.24711	-0.24711	-0.47067	4
0.49286	0.51172	0.55193	0.5829	0.68912	0.93377	0.48903	0.39253	0.60005	0.65285	0.57571	0.56863	0.56863	1.0831	4
0.52902	0.54927	0.59242	0.62567	0.73968	1.0023	0.5249	0.42133	0.64407	0.70075	0.61794	0.61035	0.61035	1.1625	4
0.29131	0.30246	0.32623	0.34453	0.40732	0.55192	0.28905	0.23201	0.35467	0.38588	0.34028	0.3361	0.3361	0.64016	4
0.90539	0.94004	1.0139	1.0708	1.2659	1.7153	0.89835	0.72108	1.1023	1.1993	1.0576	1.0446	1.0446	1.9896	4
0.65271	0.67769	0.73093	0.77195	0.91262	1.2366	0.64763	0.51984	0.79466	0.86459	0.76242	0.75306	0.75306	1.4343	4

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.40629	0.42184	0.45498	0.48051	0.56807	0.76975	0.40313	0.32358	0.49465	0.53818	0.47458	0.46875	0.46875	0.89282	4
0.72474	0.75248	0.8116	0.85715	1.0133	1.3731	0.71911	0.57721	0.88236	0.96001	0.84657	0.83616	0.83616	1.5926	4
0.64403	0.66868	0.72121	0.76169	0.90049	1.2202	0.63902	0.51293	0.78409	0.85309	0.75229	0.74304	0.74304	1.4153	4
0.88908	0.92311	0.99563	1.0515	1.2431	1.6844	0.88217	0.70809	1.0824	1.1777	1.0385	1.0258	1.0258	1.9538	4
-0.051092	-0.053048	-0.057215	-0.060426	-0.071437	-0.096799	-0.050695	-0.040692	-0.062204	-0.067678	-0.05968	-0.058947	-0.058947	-0.11228	4
-0.26499	-0.27513	-0.29675	-0.3134	-0.37051	-0.50205	-0.26293	-0.21105	-0.32262	-0.35101	-0.30954	-0.30573	-0.30573	-0.58232	4
0.060365	0.062675	0.067599	0.071393	0.084402	0.11437	0.059895	0.048076	0.073492	0.07996	0.070511	0.069645	0.069645	0.13265	4
0.39552	0.41066	0.44292	0.46778	0.55302	0.74935	0.39245	0.31501	0.48154	0.52392	0.46201	0.45633	0.45633	0.86916	4
0.64881	0.67364	0.72657	0.76734	0.90717	1.2292	0.64376	0.51673	0.78991	0.85942	0.75787	0.74856	0.74856	1.4258	4
0.43047	0.44694	0.48205	0.50911	0.60188	0.81555	0.42712	0.34284	0.52408	0.5702	0.50282	0.49664	0.49664	0.94595	4
0.064389	0.066853	0.072106	0.076152	0.090029	0.12199	0.063888	0.051281	0.078392	0.085291	0.075212	0.074288	0.074288	0.14149	4
-0.40945	-0.42512	-0.45852	-0.48425	-0.57249	-0.77573	-0.40626	-0.3261	-0.49849	-0.54236	-0.47827	-0.47239	-0.47239	-0.89976	4
0.39872	0.41398	0.4465	0.47156	0.55749	0.7554	0.39562	0.31755	0.48543	0.52815	0.46574	0.46002	0.46002	0.87618	4
0.31185	0.32378	0.34922	0.36882	0.43603	0.59083	0.30942	0.24837	0.37967	0.41308	0.36427	0.35979	0.35979	0.68529	4
0.328	0.34056	0.36731	0.38793	0.45862	0.62143	0.32545	0.26123	0.39934	0.43448	0.38314	0.37843	0.37843	0.72079	4
0.33578	0.34863	0.37602	0.39712	0.46949	0.63616	0.33317	0.26742	0.4088	0.44478	0.39222	0.3874	0.3874	0.73787	4
0.87387	0.90731	0.9786	1.0335	1.2218	1.6556	0.86707	0.69598	1.0639	1.1575	1.0208	1.0082	1.0082	1.9203	4
0.85286	0.8855	0.95507	1.0087	1.1925	1.6158	0.84623	0.67925	1.0383	1.1297	0.99622	0.98398	0.98398	1.8742	4
0.84517	0.87752	0.94646	0.99958	1.1817	1.6013	0.8386	0.67312	1.029	1.1195	0.98724	0.97511	0.97511	1.8573	4
0.36355	0.37746	0.40712	0.42997	0.50832	0.68878	0.36072	0.28954	0.44262	0.48157	0.42466	0.41944	0.41944	0.79891	4
0.64474	0.92211	0.99456	1.0504	1.1111	1.4169	0.8445	0.8132	0.93109	0.78428	1.0011	0.93235	1.0247	1.279	5
0.071643	0.10246	0.11051	0.11672	0.12346	0.15745	0.093839	0.090362	0.10346	0.087148	0.11124	0.1036	0.11386	0.14212	5
0.32126	0.45947	0.49557	0.52338	0.55362	0.70604	0.4208	0.40521	0.46395	0.39079	0.49883	0.46457	0.51057	0.63729	5
0.15125	0.21632	0.23332	0.24641	0.26065	0.33241	0.19812	0.19077	0.21843	0.18399	0.23485	0.21873	0.24038	0.30004	5
0.61193	0.87519	0.94395	0.99692	1.0545	1.3448	0.80152	0.77182	0.88371	0.74437	0.95015	0.8849	0.97252	1.2139	5
0.48592	0.69496	0.74956	0.79162	0.83736	1.0679	0.63646	0.61288	0.70172	0.59108	0.75449	0.70267	0.77224	0.96391	5
-0.12581	-0.17994	-0.19407	-0.20496	-0.21681	-0.27649	-0.16479	-0.15868	-0.18169	-0.15304	-0.19535	-0.18193	-0.19995	-0.24957	5
0.57737	0.82575	0.89062	0.94061	0.99495	1.2689	0.75624	0.72822	0.83379	0.70232	0.89648	0.83492	0.91758	1.1453	5
0.55162	0.78893	0.85092	0.89867	0.95059	1.2123	0.72253	0.69575	0.79662	0.67101	0.85651	0.79769	0.87667	1.0943	5
0.50569	0.72324	0.78006	0.82384	0.87144	1.1114	0.66236	0.63782	0.73028	0.61513	0.78519	0.73127	0.80367	1.0031	5
0.12178	0.17417	0.18786	0.1984	0.20986	0.26764	0.15951	0.1536	0.17587	0.14814	0.18909	0.17611	0.19354	0.24158	5

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.65333	0.93439	1.0078	1.0644	1.1259	1.4358	0.85574	0.82403	0.94349	0.79472	1.0144	0.94476	1.0383	1.296	5
0.62966	0.90055	0.9713	1.0258	1.0851	1.3838	0.82475	0.79419	0.90932	0.76594	0.97769	0.91054	1.0007	1.2491	5
0.53583	0.76635	0.82656	0.87295	0.92338	1.1776	0.70184	0.67584	0.77381	0.6518	0.83199	0.77486	0.85157	1.0629	5
0.59474	0.85059	0.91742	0.96891	1.0249	1.307	0.779	0.75013	0.85888	0.72345	0.92345	0.86003	0.94519	1.1798	5
-0.098471	-0.14083	-0.1519	-0.16042	-0.16969	-0.21641	-0.12898	-0.1242	-0.1422	-0.11978	-0.1529	-0.1424	-0.1565	-0.19534	5
0.25097	0.35894	0.38714	0.40887	0.43249	0.55156	0.32873	0.31655	0.36243	0.30529	0.38968	0.36292	0.39886	0.49785	5
-0.1084	-0.15503	-0.16721	-0.1766	-0.1868	-0.23823	-0.14198	-0.13672	-0.15654	-0.13186	-0.16831	-0.15675	-0.17227	-0.21503	5
0.61506	0.87965	0.94876	1.002	1.0599	1.3517	0.80561	0.77576	0.88822	0.74817	0.955	0.88942	0.97748	1.2201	5
0.64174	0.91781	0.98992	1.0455	1.1059	1.4103	0.84056	0.80941	0.92675	0.78062	0.99643	0.928	1.0199	1.273	5
0.65483	0.93654	1.0101	1.0668	1.1284	1.4391	0.85771	0.82593	0.94566	0.79655	1.0168	0.94694	1.0407	1.299	5
0.35415	0.50651	0.54631	0.57697	0.6103	0.77832	0.46388	0.44669	0.51145	0.4308	0.5499	0.51214	0.56284	0.70254	5
0.39944	0.57128	0.61617	0.65075	0.68835	0.87785	0.5232	0.50381	0.57685	0.48589	0.62022	0.57762	0.63481	0.79238	5
0.042303	0.060502	0.065256	0.068918	0.0729	0.09297	0.05541	0.053356	0.061091	0.051459	0.065685	0.061174	0.067231	0.083917	5
0.52378	0.74911	0.80796	0.85331	0.90261	1.1511	0.68605	0.66063	0.7564	0.63714	0.81327	0.75742	0.83242	1.039	5
0.57711	0.82538	0.89022	0.94019	0.99451	1.2683	0.75591	0.7279	0.83342	0.70201	0.89608	0.83454	0.91717	1.1448	5
0.34521	0.49371	0.5325	0.56239	0.59488	0.75866	0.45216	0.4354	0.49852	0.41992	0.53601	0.4992	0.54862	0.68479	5
0.034854	0.049849	0.053765	0.056783	0.060063	0.076599	0.045653	0.043961	0.050334	0.042398	0.054119	0.050402	0.055392	0.069141	5
0.40942	0.58555	0.63156	0.667	0.70554	0.89978	0.53627	0.51639	0.59126	0.49803	0.63571	0.59205	0.65067	0.81217	5
0.61992	0.88661	0.95627	1.0099	1.0683	1.3624	0.81198	0.7819	0.89525	0.75409	0.96256	0.89645	0.98521	1.2297	5
0.57398	0.8209	0.8854	0.93509	0.98912	1.2614	0.75181	0.72395	0.8289	0.6982	0.89122	0.83002	0.9122	1.1386	5
0.57405	0.82101	0.88551	0.93521	0.98925	1.2616	0.75191	0.72404	0.82901	0.69829	0.89134	0.83013	0.91232	1.1388	5
0.097993	0.14015	0.15116	0.15964	0.16887	0.21536	0.12835	0.1236	0.14151	0.1192	0.15215	0.14171	0.15574	0.19439	5
0.17826	0.25495	0.27498	0.29041	0.30719	0.39176	0.23349	0.22484	0.25743	0.21684	0.27678	0.25778	0.2833	0.35361	5
0.61032	0.87289	0.94146	0.9943	1.0518	1.3413	0.79941	0.76979	0.88139	0.74241	0.94766	0.88258	0.96996	1.2107	5
0.3937	0.56308	0.60732	0.6414	0.67846	0.86524	0.51568	0.49657	0.56856	0.47891	0.61131	0.56933	0.6257	0.78099	5
0.38487	0.55044	0.59368	0.627	0.66323	0.84582	0.50411	0.48543	0.5558	0.46816	0.59759	0.55655	0.61165	0.76346	5
0.10259	0.14672	0.15825	0.16713	0.17678	0.22545	0.13437	0.12939	0.14815	0.12479	0.15929	0.14835	0.16303	0.2035	5
0.64944	0.92883	1.0018	1.058	1.1192	1.4273	0.85065	0.81913	0.93787	0.78999	1.0084	0.93914	1.0321	1.2883	5
0.64911	0.92836	1.0013	1.0575	1.1186	1.4265	0.85022	0.81871	0.9374	0.78959	1.0079	0.93867	1.0316	1.2876	5
-0.21646	-0.30958	-0.3339	-0.35264	-0.37302	-0.47571	-0.28352	-0.27302	-0.31259	-0.26331	-0.3361	-0.31302	-0.34401	-0.42939	5
0.38341	0.54836	0.59144	0.62463	0.66072	0.84263	0.5022	0.48359	0.5537	0.46639	0.59533	0.55445	0.60934	0.76058	5

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/ min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.47144	0.67425	0.72722	0.76803	0.81241	1.0361	0.6175	0.59461	0.68081	0.57346	0.732	0.68173	0.74923	0.93519	5
-0.035916	-0.051367	-0.055402	-0.058512	-0.061892	-0.078932	-0.047043	-0.0453	-0.051867	-0.043689	-0.055767	-0.051937	-0.057079	-0.071246	5
0.24037	0.34377	0.37078	0.39159	0.41422	0.52825	0.31484	0.30317	0.34712	0.29239	0.37322	0.34759	0.382	0.47682	5
0.58218	0.83263	0.89804	0.94844	1.0032	1.2794	0.76254	0.73429	0.84074	0.70817	0.90395	0.84187	0.92522	1.1549	5
-0.011784	-0.016854	-0.018178	-0.019198	-0.020308	-0.025898	-0.015435	-0.014863	-0.017018	-0.014335	-0.018298	-0.017041	-0.018728	-0.023377	5
0.39867	0.57018	0.61497	0.64949	0.68701	0.87615	0.52218	0.50284	0.57573	0.48495	0.61902	0.57651	0.63359	0.79084	5
0.6238	0.89216	0.96226	1.0163	1.075	1.3709	0.81707	0.78679	0.90085	0.75881	0.96858	0.90207	0.99138	1.2374	5
-0.059145	-0.084589	-0.091235	-0.096355	-0.10192	-0.12998	-0.077469	-0.074599	-0.085413	-0.071945	-0.091835	-0.085528	-0.093996	-0.11733	5

Appendix H

Testing Dataset of the Practical Experiment for the Multi-Layer Perceptron

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.069062	0.12343	0.11201	0.071234	0.059837	0.042792	0,11766	0.13245	0.080662	0.099728	0.093396	0.095784	0.095784	0.040044	1
-0.57922	-1.0352	-0.93944	-0.59743	-0.50185	-0.35889	-0.98677	-1.1109	-0.67651	-0.83641	-0.7833	-0.80333	-0.80333	-0.33585	1
0.54707	0.97771	0.8873	0.56428	0.47399	0.33897	0.93201	1.0492	0.63896	0.78999	0.73983	0.75874	0.75874	0.31721	1
0.53011	0.9474	0.8598	0.54678	0.4593	0.32847	0.90312	1.0167	0.61915	0.7655	0.71689	0.73522	0.73522	0.30738	1
0.6222	1.112	1.0092	0.64177	0.53908	0.38552	1.06	1.1933	0.7267	0.89847	0.84143	0.86294	0.86294	0.36077	1
0.21746	0.38863	0.3527	0.22429	0.18841	0.13474	0.37046	0.41705	0.25398	0.31401	0.29408	0.30159	0.30159	0.12609	1
-0.34621	-0.61873	-0.56152	-0.3571	-0.29996	-0.21452	-0.58981	-0.66398	-0.40436	-0.49994	-0.46819	-0.48017	-0.48017	-0.20074	1
0.65363	1.1681	1.0601	0.67419	0.56632	0.405	1.1135	1.2536	0.76342	0.94386	0.88393	0.90653	0.90653	0.379	1
0.040125	0.071709	0.065078	0.041386	0.034765	0.024862	0.068357	0.076953	0.046864	0.057941	0.054262	0.055649	0.055649	0.023265	1
-1.3015	-2.3261	-2.111	-1.3425	-1.1277	-0.80645	-2.2173	-2.4962	-1.5202	-1.8795	-1.7601	-1.8051	-1.8051	-0.75467	1
-0.13795	-0.24655	-0.22375	-0.14229	-0.11953	-0.085478	-0.23502	-0.26457	-0.16112	-0.19921	-0.18656	-0.19133	-0.19133	-0.07999	1
0.44324	0.79215	0.7189	0.45718	0.38403	0.27464	0.75512	0.85007	0.51769	0.64006	0.59942	0.61474	0.61474	0.25701	1
-0.56267	-1.0056	-0.9126	-0.58036	-0.4875	-0.34864	-0.95858	-1.0791	-0.65718	-0.81251	-0.76092	-0.78037	-0.78037	-0.32625	1
-0.11611	-0.20751	-0.18833	-0.11976	-0.1006	-0.071946	-0.19781	-0.22269	-0.13562	-0.16767	-0.15703	-0.16104	-0.16104	-0.067326	1
0.59119	1.0566	0.95887	0.60979	0.51222	0.36631	1.0072	1.1338	0.69049	0.8537	0.7995	0.81994	0.81994	0.34279	1
-0.77566	-1.3862	-1.2581	-0.80006	-0.67205	-0.48061	-1.3214	-1.4876	-0.90594	-1.1201	-1.049	-1.0758	-1.0758	-0.44975	1
0.6615	1.1822	1.0729	0.6823	0.57313	0.40987	1.1269	1.2687	0.77261	0.95522	0.89457	0.91744	0.91744	0.38356	1
-0.18531	-0.33118	-0.30056	-0.19114	-0.16056	-0.11482	-0.3157	-0.3554	-0.21644	-0.26759	-0.2506	-0.25701	-0.25701	-0.10745	1
0.21764	0.38896	0.35299	0.22449	0.18857	0.13485	0.37078	0.4174	0.2542	0.31428	0.29432	0.30185	0.30185	0.1262	1
0.36605	0.6542	0.59371	0.37756	0.31715	0.22681	0.62362	0.70203	0.42754	0.52859	0.49503	0.50769	0.50769	0.21225	1
-1.2035	-2.1508	-1.9519	-1.2413	-1.0427	-0.74568	-2.0502	-2.3081	-1.4056	-1.7378	-1.6275	-1.6691	-1.6691	-0.6978	1
0.031489	0.056275	0.051072	0.032479	0.027282	0.019511	0.053645	0.06039	0.036777	0.04547	0.042583	0.043672	0.043672	0.018258	1
0.12247	0.21887	0.19863	0.12632	0.10611	0.075883	0.20864	0.23488	0.14304	0.17685	0.16562	0.16985	0.16985	0.071011	1
0.58669	1.0485	0.95156	0.60514	0.50832	0.36352	0.9995	1.1252	0.68523	0.8472	0.79341	0.81369	0.81369	0.34018	1
0.40944	0.73173	0.66407	0.42231	0.35474	0.25369	0.69753	0.78524	0.47821	0.59124	0.5537	0.56786	0.56786	0.23741	1
0.66372	1.1862	1.0765	0.6846	0.57506	0.41125	1.1307	1.2729	0.7752	0.95843	0.89758	0.92053	0.92053	0.38485	1

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.39838	0.71196	0.64613	0.4109	0.34516	0.24684	0.67868	0.76402	0.46529	0.57527	0.53874	0.55251	0.55251	0.23099	1
0.25313	0.45238	0.41055	0.26109	0.21931	0.15684	0.43123	0.48546	0.29564	0.36552	0.34231	0.35106	0.35106	0.14677	1
0.032441	0.057977	0.052616	0.033461	0.028107	0.020101	0.055267	0.062217	0.03789	0.046845	0.043871	0.044993	0.044993	0.01881	1
0.4965	0.88732	0.80528	0.51211	0.43018	0.30764	0.84585	0.95221	0.57989	0.71696	0.67144	0.6886	0.6886	0.28789	1
-0.36497	-0.65226	-0.59195	-0.37645	-0.31621	-0.22614	-0.62177	-0.69995	-0.42627	-0.52702	-0.49356	-0.50618	-0.50618	-0.21162	1
-1.3472	-2.4076	-2.185	-1.3895	-1.1672	-0.83471	-2.295	-2.5836	-1.5734	-1.9453	-1.8218	-1.8684	-1.8684	-0.78112	1
0.0073525	0.01314	0.011925	0.0075837	0.0063703	0.0045557	0.012526	0.014101	0.0085875	0.010617	0.0099431	0.010197	0.010197	0.0042632	1
0.68954	1.2323	1.1184	0.71122	0.59743	0.42725	1.1747	1.3224	0.80536	0.99571	0.93249	0.95634	0.95634	0.39982	1
0.50281	0.89859	0.81551	0.51862	0.43564	0.31155	0.85659	0.9643	0.58726	0.72606	0.67997	0.69735	0.69735	0.29154	1
0.33798	0.60402	0.54817	0.34861	0.29283	0.20942	0.57579	0.64819	0.39475	0.48805	0.45706	0.46875	0.46875	0.19597	1
0.45205	0.80788	0.73318	0.46626	0.39166	0.28009	0.77012	0.86695	0.52797	0.65276	0.61132	0.62695	0.62695	0.26211	1
0.45607	0.81507	0.7397	0.47041	0.39515	0.28259	0.77697	0.87467	0.53267	0.65858	0.61676	0.63253	0.63253	0.26444	1
-0.23844	-0.42612	-0.38672	-0.24593	-0.20658	-0.14774	-0.40621	-0.45728	-0.27848	-0.34431	-0.32245	-0.33069	-0.33069	-0.13825	1
0.47848	0.85512	0.77605	0.49353	0.41456	0.29647	0.81515	0.91765	0.55885	0.69094	0.64707	0.66361	0.66361	0.27744	1
0.66276	1.1845	1.0749	0.6836	0.57422	0.41065	1.1291	1.2711	0.77408	0.95704	0.89627	0.91919	0.91919	0.38429	1
0.61833	1.1051	1.0029	0.63778	0.53573	0.38313	1.0534	1.1859	0.72219	0.89289	0.8362	0.85758	0.85758	0.35853	1
-1.6505	-2.9497	-2.677	-1.7024	-1.43	-1.0227	-2.8118	-3.1654	-1.9277	-2.3834	-2.232	-2.2891	-2.2891	-0.95701	1
0.63963	1.1431	1.0374	0.65974	0.55419	0.39632	1.0897	1.2267	0.74706	0.92364	0.865	0.88711	0.88711	0.37088	1
0.13731	0.2454	0.22271	0.14163	0.11897	0.08508	0.23393	0.26334	0.16037	0.19828	0.18569	0.19044	0.19044	0.079617	1
0.58917	1.0529	0.95559	0.6077	0.51047	0.36506	1.0037	1.1299	0.68813	0.85078	0.79676	0.81713	0.81713	0.34162	1
0.62522	1.1174	1.014	0.64488	0.5417	0.38739	1.0651	1.1991	0.73023	0.90283	0.84551	0.86712	0.86712	0.36252	1
0.65539	1.1713	1.063	0.676	0.56784	0.40609	1.1165	1.2569	0.76547	0.94641	0.88632	0.90898	0.90898	0.38002	1
-0.43239	-0.77274	-0.70129	-0.44598	-0.37463	-0.26791	-0.73662	-0.82925	-0.50501	-0.62438	-0.58473	-0.59968	-0.59968	-0.25071	1
0.65458	1.1698	1.0617	0.67517	0.56714	0.40559	1.1152	1.2554	0.76453	0.94524	0.88522	0.90785	0.90785	0.37955	1
1.3055	0.78789	0.80089	0.94734	0.94313	0.55911	0.91068	0.94879	0.97521	0.88419	0.91226	0.95745	0.88668	0.50603	2
-3.1373	-1.8935	-1.9247	-2.2767	-2.2665	-1.3436	-2.1885	-2.2801	-2.3436	-2.1249	-2.1923	-2.3009	-2.1309	-1.2161	2
-1.2355	-0.74564	-0.75795	-0.89655	-0.89256	-0.52913	-0.86185	-0.89792	-0.92292	-0.83678	-0.86334	-0.90612	-0.83914	-0.4789	2
-3.7367	-2.2552	-2.2924	-2.7116	-2.6995	-1.6003	-2.6067	-2.7157	-2.7913	-2.5308	-2.6112	-2.7405	-2.538	-1.4484	2
1.4746	0.88997	0.90466	1.0701	1.0653	0.63154	1.0287	1.0717	1.1016	0.99874	1.0304	1.0815	1.0016	0.57159	2
1.2307	0.74275	0.75501	0.89307	0.8891	0.52708	0.85851	0.89444	0.91934	0.83353	0.85999	0.9026	0.83589	0.47704	2
1.1157	0.67338	0.68449	0.80965	0.80606	0.47785	0.77832	0.8109	0.83347	0.75568	0.77967	0.8183	0.75781	0.43248	2

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.80482	0.48572	0.49374	0.58403	0.58143	0.34468	0.56142	0.58492	0.6012	0.54509	0.5624	0.59026	0.54663	0.31196	2
1.0989	0.66321	0.67416	0.79743	0.79389	0.47063	0.76657	0.79866	0.82089	0.74427	0.7679	0.80594	0.74637	0.42595	2
0.11726	0.070768	0.071936	0.08509	0.084712	0.050219	0.081797	0.08522	0.087593	0.079417	0.081938	0.085998	0.079642	0.045451	2
1.0661	0.64339	0.65401	0.7736	0.77016	0.45657	0.74366	0.77479	0.79636	0.72203	0.74495	0.78186	0.72407	0.41322	2
1.4598	0.88103	0.89557	1.0593	1.0546	0.6252	1.0183	1.061	1.0905	0.98871	1.0201	1.0706	0.9915	0.56585	2
-1.0009	-0.60406	-0.61403	-0.72632	-0.72309	-0.42866	-0.69821	-0.72743	-0.74768	-0.67789	-0.69941	-0.73407	-0.67981	-0.38796	2
0.84185	0.50807	0.51646	0.6109	0.60818	0.36054	0.58726	0.61183	0.62887	0.57017	0.58827	0.61742	0.57178	0.32631	2
0.62854	0.37934	0.3856	0.45611	0.45408	0.26919	0.43846	0.45681	0.46953	0.4257	0.43922	0.46098	0.42691	0.24363	2
-0.06074	-0.036658	-0.037263	-0.044077	-0.043881	-0.026013	-0.042371	-0.044144	-0.045373	-0.041138	-0.042444	-0.044547	-0.041255	-0.023544	2
0.6638	0.40062	0.40723	0.4817	0.47956	0.28429	0.46305	0.48243	0.49586	0.44958	0.46386	0.48684	0.45085	0.2573	2
-3.6288	-2.1901	-2.2262	-2.6333	-2.6216	-1.5541	-2.5314	-2.6373	-2.7107	-2.4577	-2.5357	-2.6614	-2.4647	-1.4066	2
1.4434	0.8711	0.88548	1.0474	1.0427	0.61815	1.0069	1.049	1.0782	0.97757	1.0086	1.0586	0.98033	0.55947	2
0.68096	0.41097	0.41776	0.49414	0.49195	0.29164	0.47502	0.4949	0.50868	0.4612	0.47584	0.49942	0.4625	0.26395	2
-0.24305	-0.14669	-0.14911	-0.17638	-0.17559	-0.10409	-0.16955	-0.17665	-0.18156	-0.16462	-0.16984	-0.17826	-0.16508	-0.094212	2
0.35762	0.21583	0.21939	0.25951	0.25836	0.15316	0.24947	0.25991	0.26714	0.24221	0.2499	0.26228	0.24289	0.13862	2
-1.5305	-0.9237	-0.93895	-1.1106	-1.1057	-0.65548	-1.0677	-1.1123	-1.1433	-1.0366	-1.0695	-1.1225	-1.0395	-0.59325	2
1.475	0.89017	0.90486	1.0703	1.0656	0.63168	1.0289	1.072	1.1018	0.99896	1.0307	1.0817	1.0018	0.57171	2
0.53588	0.32342	0.32876	0.38887	0.38714	0.22951	0.37382	0.38947	0.40031	0.36295	0.37447	0.39302	0.36397	0.20772	2
0.61651	0.37208	0.37822	0.44738	0.44539	0.26404	0.43007	0.44807	0.46054	0.41756	0.43081	0.45216	0.41874	0.23897	2
0.86541	0.52229	0.53091	0.628	0.6252	0.37063	0.60369	0.62896	0.64647	0.58613	0.60474	0.6347	0.58779	0.33545	2
-0.37392	-0.22567	-0.22939	-0.27134	-0.27013	-0.16014	-0.26084	-0.27176	-0.27932	-0.25325	-0.26129	-0.27424	-0.25397	-0.14494	2
0.88003	0.53111	0.53988	0.6386	0.63576	0.37689	0.61389	0.63958	0.65738	0.59603	0.61495	0.64542	0.59771	0.34111	2
-1.2205	-0.73658	-0.74874	-0.88565	-0.88172	-0.5227	-0.85138	-0.88701	-0.9117	-0.82661	-0.85285	-0.8951	-0.82894	-0.47307	2
-0.92897	-0.56065	-0.56991	-0.67412	-0.67112	-0.39786	-0.64803	-0.67515	-0.69395	-0.62918	-0.64915	-0.68131	-0.63096	-0.36008	2
0.99206	0.59873	0.60861	0.7199	0.7167	0.42488	0.69204	0.72101	0.74108	0.67191	0.69324	0.72759	0.67381	0.38454	2
-0.065825	-0.039727	-0.040382	-0.047767	-0.047554	-0.028191	-0.045918	-0.04784	-0.049171	-0.044582	-0.045997	-0.048276	-0.044708	-0.025515	2
1.3745	0.82957	0.84326	0.99746	0.99303	0.58868	0.95886	0.99899	1.0268	0.93096	0.96052	1.0081	0.93359	0.5328	2
1.2934	0.78063	0.79351	0.93861	0.93444	0.55395	0.90228	0.94005	0.96622	0.87604	0.90385	0.94863	0.87851	0.50136	2
-0.33716	-0.20348	-0.20684	-0.24466	-0.24357	-0.1444	-0.23519	-0.24504	-0.25186	-0.22835	-0.2356	-0.24727	-0.229	-0.13069	2
1.4671	0.88542	0.90004	1.0646	1.0599	0.62832	1.0234	1.0662	1.0959	0.99364	1.0252	1.076	0.99645	0.56867	2
0.55032	0.33213	0.33761	0.39935	0.39757	0.23569	0.38389	0.39996	0.41109	0.37272	0.38456	0.40361	0.37378	0.21331	2

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
-1.0698	-0.64565	-0.6563	-0.77631	-0.77286	-0.45817	-0.74627	-0.7775	-0.79915	-0.72456	-0.74756	-0.7846	-0.72661	-0.41467	2
0.55037	0.33216	0.33764	0.39938	0.39761	0.23571	0.38393	0.4	0.41113	0.37276	0.38459	0.40365	0.37381	0.21333	2
1.38	0.83288	0.84662	1.0014	0.99699	0.59103	0.96268	1.003	1.0309	0.93467	0.96435	1.0121	0.93731	0.53492	2
0.48732	0.29411	0.29896	0.35363	0.35206	0.20871	0.33994	0.35417	0.36403	0.33005	0.34053	0.3574	0.33098	0.18889	2
1.3595	0.8205	0.83404	0.98655	0.98216	0.58225	0.94837	0.98806	1.0156	0.92078	0.95001	0.99708	0.92338	0.52697	2
0.15576	0.094006	0.095557	0.11303	0.11253	0.066709	0.10866	0.1132	0.11636	0.1055	0.10884	0.11424	0.10579	0.060376	2
0.73523	0.44373	0.45105	0.53353	0.53116	0.31488	0.51288	0.53435	0.54922	0.49796	0.51377	0.53922	0.49937	0.28499	2
0.98978	0.59735	0.60721	0.71825	0.71506	0.4239	0.69045	0.71935	0.73937	0.67036	0.69165	0.72591	0.67226	0.38365	2
-0.43	-0.25952	-0.2638	-0.31204	-0.31065	-0.18416	-0.29996	-0.31251	-0.32121	-0.29123	-0.30048	-0.31537	-0.29206	-0.16668	2
-1.9394	-1.1705	-1.1898	-1.4074	-1.4011	-0.83061	-1.3529	-1.4095	-1.4488	-1.3136	-1.3553	-1.4224	-1.3173	-0.75175	2
0.023605	0.014246	0.014481	0.017129	0.017053	0.010109	0.016466	0.017156	0.017633	0.015987	0.016495	0.017312	0.016033	0.0091497	2
-2.1363	-1.2893	-1.3106	-1.5502	-1.5433	-0.91491	-1.4902	-1.5526	-1.5958	-1.4469	-1.4928	-1.5668	-1.451	-0.82805	2
0.68055	0.53164	0.50583	0.57531	0.50116	0.41715	0.55305	0.56262	0.55604	0.53696	0.51139	0.51572	0.50814	0.39239	3
-1.7906	-1.3988	-1.3309	-1.5137	-1.3186	-1.0976	-1.4551	-1.4803	-1.463	-1.4128	-1.3455	-1.3569	-1.337	-1.0324	3
-0.99357	-0.77617	-0.73848	-0.83992	-0.73167	-0.60901	-0.80742	-0.8214	-0.81179	-0.78393	-0.7466	-0.75293	-0.74185	-0.57287	3
-1.5212	-1.1883	-1.1306	-1.286	-1.1202	-0.93242	-1.2362	-1.2576	-1.2429	-1.2002	-1.1431	-1.1528	-1.1358	-0.87709	3
-1.217	-0.95069	-0.90453	-1.0288	-0.89618	-0.74595	-0.98896	-1.0061	-0.99432	-0.96019	-0.91447	-0.92222	-0.90866	-0.70168	3
-0.47516	-0.37119	-0.35317	-0.40168	-0.34991	-0.29125	-0.38614	-0.39282	-0.38823	-0.3749	-0.35705	-0.36008	-0.35478	-0.27397	3
0.78835	0.61585	0.58595	0.66644	0.58054	0.48322	0.64064	0.65174	0.64411	0.62201	0.59239	0.59741	0.58862	0.45454	3
1.2585	0.98312	0.93539	1.0639	0.92675	0.7714	1.0227	1.0404	1.0282	0.99295	0.94567	0.95368	0.93966	0.72562	3
0.903	0.70542	0.67117	0.76336	0.66497	0.5535	0.73382	0.74652	0.73779	0.71247	0.67854	0.68429	0.67423	0.52065	3
-1.2365	-0.96591	-0.91902	-1.0453	-0.91053	-0.7579	-1.0048	-1.0222	-1.0102	-0.97557	-0.92912	-0.93699	-0.92321	-0.71292	3
-1.4378	-1.1232	-1.0687	-1.2155	-1.0588	-0.88131	-1.1684	-1.1887	-1.1748	-1.1344	-1.0804	-1.0896	-1.0735	-0.82901	3
1.1943	0.933	0.8877	1.0096	0.87951	0.73207	0.97056	0.98737	0.97582	0.94233	0.89746	0.90506	0.89175	0.68862	3
1.1809	0.9225	0.87771	0.99828	0.86961	0.72384	0.95964	0.97626	0.96484	0.93173	0.88736	0.89488	0.88172	0.68088	3
-0.83553	-0.65271	-0.62102	-0.70633	-0.61529	-0.51215	-0.67899	-0.69075	-0.68267	-0.65924	-0.62785	-0.63317	-0.62386	-0.48175	3
1.1856	0.92616	0.8812	1.0022	0.87306	0.72671	0.96345	0.98013	0.96867	0.93542	0.89088	0.89843	0.88522	0.68358	3
-0.64418	-0.50323	-0.47879	-0.54456	-0.47437	-0.39485	-0.52349	-0.53255	-0.52632	-0.50826	-0.48406	-0.48816	-0.48098	-0.37142	3
0.9641	0.75315	0.71658	0.81502	0.70997	0.59095	0.78347	0.79704	0.78772	0.76068	0.72446	0.7306	0.71985	0.55588	3
0.17314	0.13525	0.12869	0.14636	0.1275	0.10613	0.1407	0.14313	0.14146	0.13661	0.1301	0.1312	0.12927	0.099827	3
-1.8148	-1.4177	-1.3488	-1.5341	-1.3364	-1.1124	-1.4748	-1.5003	-1.4827	-1.4319	-1.3637	-1.3752	-1.355	-1.0464	3

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
1.0498	0.82006	0.78024	0.88742	0.77304	0.64345	0.85308	0.86785	0.8577	0.82826	0.78882	0.7955	0.78381	0.60527	3
-0.7462	-0.58293	-0.55463	-0.63081	-0.54951	-0.45739	-0.6064	-0.6169	-0.60968	-0.58876	-0.56072	-0.56547	-0.55716	-0.43025	3
-1.7125	-1.3378	-1.2729	-1.4477	-1.2611	-1.0497	-1.3917	-1.4158	-1.3992	-1.3512	-1.2868	-1.2978	-1.2787	-0.98741	3
-3.0052	-2.3477	-2.2337	-2.5405	-2.2131	-1.8421	-2.4422	-2.4845	-2.4554	-2.3711	-2.2582	-2.2774	-2.2439	-1.7327	3
-2.5664	-2.0048	-1.9075	-2.1695	-1.8899	-1.5731	-2.0855	-2.1216	-2.0968	-2.0249	-1.9284	-1.9448	-1.9162	-1.4797	3
-1.4177	-1.1075	-1.0537	-1.1984	-1.044	-0.86897	-1.1521	-1.172	-1.1583	-1.1185	-1.0653	-1.0743	-1.0585	-0.8174	3
1.2501	0.97656	0.92915	1.0568	0.92057	0.76625	1.0159	1.0335	1.0214	0.98633	0.93936	0.94732	0.93339	0.72078	3
0.20102	0.15703	0.14941	0.16993	0.14803	0.12322	0.16336	0.16619	0.16424	0.15861	0.15105	0.15233	0.15009	0.1159	3
0.16281	0.12718	0.12101	0.13763	0.11989	0.099793	0.1323	0.13459	0.13302	0.12845	0.12234	0.12337	0.12156	0.09387	3
1.234	0.96402	0.91722	1.0432	0.90875	0.75641	1.0028	1.0202	1.0083	0.97366	0.9273	0.93516	0.9214	0.71152	3
0.36341	0.2839	0.27011	0.30722	0.26762	0.22276	0.29533	0.30044	0.29693	0.28674	0.27308	0.2754	0.27135	0.20954	3
-0.44628	-0.34863	-0.33171	-0.37727	-0.32865	-0.27355	-0.36267	-0.36895	-0.36463	-0.35212	-0.33535	-0.33819	-0.33322	-0.25732	3
0.78365	0.61218	0.58246	0.66246	0.57708	0.48034	0.63683	0.64785	0.64027	0.6183	0.58886	0.59385	0.58511	0.45183	3
-3.3456	-2.6136	-2.4867	-2.8282	-2.4637	-2.0507	-2.7188	-2.7659	-2.7335	-2.6397	-2.514	-2.5353	-2.498	-1.929	3
1.1992	0.93683	0.89135	1.0138	0.88312	0.73508	0.97455	0.99142	0.97982	0.9462	0.90114	0.90878	0.89541	0.69145	3
0.16224	0.12674	0.12059	0.13715	0.11948	0.099448	0.13185	0.13413	0.13256	0.12801	0.12192	0.12295	0.12114	0.093546	3
1.0499	0.82017	0.78035	0.88754	0.77314	0.64354	0.85319	0.86796	0.85781	0.82837	0.78892	0.79561	0.78391	0.60535	3
0.75947	0.59329	0.56449	0.64203	0.55928	0.46552	0.61718	0.62786	0.62052	0.59922	0.57069	0.57553	0.56706	0.43789	3
-0.78767	-0.61532	-0.58545	-0.66587	-0.58004	-0.48281	-0.6401	-0.65118	-0.64356	-0.62147	-0.59188	-0.5969	-0.58812	-0.45415	3
0.56908	0.44456	0.42298	0.48108	0.41908	0.34882	0.46246	0.47047	0.46497	0.44901	0.42763	0.43125	0.42491	0.32812	3
0.049244	0.038469	0.036601	0.041629	0.036264	0.030185	0.040018	0.040711	0.040235	0.038854	0.037004	0.037317	0.036768	0.028393	3
-0.24993	-0.19524	-0.18576	-0.21128	-0.18405	-0.15319	-0.2031	-0.20662	-0.2042	-0.19719	-0.1878	-0.18939	-0.18661	-0.1441	3
-1.4177	-1.1075	-1.0537	-1.1985	-1.044	-0.86899	-1.1521	-1.172	-1.1583	-1.1186	-1.0653	-1.0743	-1.0585	-0.81742	3
0.63816	0.49853	0.47432	0.53948	0.46994	0.39116	0.5186	0.52758	0.52141	0.50351	0.47953	0.4836	0.47649	0.36795	3
1.0697	0.83568	0.7951	0.90432	0.78777	0.65571	0.86932	0.88437	0.87403	0.84403	0.80384	0.81065	0.79873	0.61679	3
0.99638	0.77836	0.74057	0.8423	0.73374	0.61074	0.8097	0.82372	0.81409	0.78615	0.74871	0.75506	0.74395	0.57449	3
-2.7958	-2.184	-2.078	-2.3634	-2.0588	-1.7137	-2.272	-2.3113	-2.2843	-2.2059	-2.1008	-2.1186	-2.0875	-1.612	3
0.77225	0.60328	0.57399	0.65283	0.56869	0.47336	0.62756	0.63843	0.63096	0.60931	0.58029	0.58521	0.57661	0.44526	3
-0.42237	-0.32996	-0.31394	-0.35706	-0.31104	-0.2589	-0.34324	-0.34918	-0.3451	-0.33325	-0.31739	-0.32008	-0.31537	-0.24353	3
0.20777	0.16231	0.15443	0.17564	0.153	0.12735	0.16884	0.17176	0.16976	0.16393	0.15612	0.15745	0.15513	0.11979	3
1.2335	0.96363	0.91684	1.0428	0.90838	0.7561	1.0024	1.0198	1.0079	0.97326	0.92692	0.93477	0.92103	0.71123	3

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
-0.11541	-0.11983	-0.12924	-0.1365	-0.16137	-0.21866	-0.11451	-0.091918	-0.14051	-0.15288	-0.13481	-0.13316	-0.13316	-0.25362	4
0.64875	0.67358	0.7265	0.76727	0.90708	1.2291	0.6437	0.51668	0.78984	0.85934	0.7578	0.74849	0.74849	1.4256	4
-0.35196	-0.36543	-0.39414	-0.41626	-0.49211	-0.66682	-0.34922	-0.28031	-0.4285	-0.46621	-0.41112	-0.40607	-0.40607	-0.77343	4
0.38366	0.39835	0.42964	0.45375	0.53644	0.72688	0.38068	0.30556	0.4671	0.5082	0.44815	0.44265	0.44265	0.8431	4
0.76775	0.79714	0.85976	0.90802	1.0735	1.4546	0.76178	0.61146	0.93472	1.017	0.89681	0.88579	0.88579	1.6871	4
0.90127	0.93576	1.0093	1.0659	1.2602	1.7075	0.89426	0.7178	1.0973	1.1938	1.0528	1.0398	1.0398	1.9805	4
0.71691	0.74435	0.80283	0.84789	1.0024	1.3583	0.71134	0.57097	0.87282	0.94963	0.83742	0.82713	0.82713	1.5754	4
0.65816	0.68335	0.73704	0.7784	0.92024	1.2469	0.65304	0.52418	0.80129	0.87181	0.76879	0.75934	0.75934	1.4463	4
0.18436	0.19141	0.20645	0.21804	0.25777	0.34928	0.18292	0.14683	0.22445	0.2442	0.21535	0.2127	0.2127	0.40512	4
0.2353	0.2443	0.2635	0.27828	0.32899	0.44579	0.23347	0.1874	0.28647	0.31168	0.27485	0.27147	0.27147	0.51706	4
-0.73169	-0.75969	-0.81938	-0.86536	-1.023	-1.3862	-0.726	-0.58274	-0.89081	-0.9692	-0.85468	-0.84418	-0.84418	-1.6079	4
-1.9123	-1.9855	-2.1415	-2.2617	-2.6738	-3.6231	-1.8975	-1.523	-2.3282	-2.5331	-2.2338	-2.2063	-2.2063	-4.2024	4
0.89982	0.93426	1.0077	1.0642	1.2581	1.7048	0.89282	0.71665	1.0955	1.1919	1.0511	1.0382	1.0382	1.9774	4
-1.5691	-1.6292	-1.7572	-1.8558	-2.1939	-2.9728	-1.5569	-1.2497	-1.9103	-2.0785	-1.8329	-1.8103	-1.8103	-3.4481	4
0.89448	0.92871	1.0017	1.0579	1.2507	1.6947	0.88752	0.71239	1.089	1.1848	1.0448	1.032	1.032	1.9656	4
-0.25076	-0.26036	-0.28081	-0.29657	-0.35061	-0.47509	-0.24881	-0.19971	-0.3053	-0.33216	-0.29291	-0.28931	-0.28931	-0.55105	4
0.68395	0.71012	0.76592	0.8089	0.9563	1.2958	0.67863	0.54472	0.83269	0.90597	0.79891	0.7891	0.7891	1.503	4
-1.9411	-2.0154	-2.1738	-2.2958	-2.7141	-3.6776	-1.926	-1.546	-2.3633	-2.5712	-2.2674	-2.2395	-2.2395	-4.2656	4
0.20003	0.20769	0.224	0.23658	0.27969	0.37898	0.19848	0.15931	0.24353	0.26497	0.23366	0.23078	0.23078	0.43957	4
0.84742	0.87985	0.94898	1.0022	1.1849	1.6055	0.84083	0.67491	1.0317	1.1225	0.98986	0.9777	0.9777	1.8622	4
0.46411	0.48187	0.51973	0.5489	0.64892	0.8793	0.4605	0.36963	0.56504	0.61477	0.54212	0.53546	0.53546	1.0199	4
0.22911	0.23788	0.25657	0.27097	0.32035	0.43408	0.22733	0.18247	0.27894	0.30349	0.26763	0.26434	0.26434	0.50348	4
0.49045	0.50922	0.54922	0.58005	0.68574	0.92919	0.48663	0.39061	0.59711	0.64965	0.57289	0.56585	0.56585	1.0778	4
0.82535	0.85694	0.92427	0.97614	1.154	1.5637	0.81893	0.65734	1.0048	1.0933	0.96409	0.95224	0.95224	1.8137	4
-0.014616	-0.015175	-0.016367	-0.017286	-0.020436	-0.027691	-0.014502	-0.01164	-0.017794	-0.01936	-0.017073	-0.016863	-0.016863	-0.032118	4
0.62704	0.65104	0.70219	0.7416	0.87673	1.188	0.62216	0.4994	0.76341	0.83059	0.73244	0.72344	0.72344	1.3779	4
-1.3111	-1.3612	-1.4682	-1.5506	-1.8331	-2.4839	-1.3009	-1.0442	-1.5962	-1.7366	-1.5314	-1.5126	-1.5126	-2.881	4
0.89712	0.93146	1.0046	1.061	1.2544	1.6997	0.89015	0.7145	1.0922	1.1883	1.0479	1.035	1.035	1.9714	4
-1.1568	-1.2011	-1.2954	-1.3681	-1.6174	-2.1916	-1.1478	-0.92131	-1.4084	-1.5323	-1.3512	-1.3346	-1.3346	-2.5421	4
0.19207	0.19943	0.21509	0.22716	0.26856	0.3639	0.19058	0.15297	0.23385	0.25442	0.22436	0.2216	0.2216	0.42208	4
0.062896	0.065303	0.070434	0.074387	0.087942	0.11916	0.062407	0.050092	0.076575	0.083313	0.073468	0.072566	0.072566	0.13821	4

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.36127	0.37509	0.40456	0.42727	0.50512	0.68445	0.35846	0.28772	0.43983	0.47854	0.42199	0.41681	0.41681	0.79388	4
0.8083	0.83924	0.90517	0.95597	1.1302	1.5314	0.80202	0.64376	0.98409	1.0707	0.94417	0.93257	0.93257	1.7762	4
0.63957	0.66405	0.71622	0.75641	0.89425	1.2117	0.63459	0.50937	0.77866	0.84718	0.74707	0.73789	0.73789	1.4055	4
0.55186	0.57298	0.618	0.65268	0.77161	1.0455	0.54757	0.43952	0.67188	0.731	0.64462	0.6367	0.6367	1.2127	4
-0.71841	-0.7459	-0.80451	-0.84966	-1.0045	-1.3611	-0.71282	-0.57217	-0.87465	-0.95162	-0.83917	-0.82886	-0.82886	-1.5787	4
0.82239	0.85386	0.92095	0.97263	1.1499	1.5581	0.81599	0.65498	1.0012	1.0893	0.96062	0.94882	0.94882	1.8072	4
0.048806	0.050674	0.054656	0.057723	0.068241	0.092468	0.048427	0.038871	0.059421	0.06465	0.05701	0.05631	0.05631	0.10725	4
0.81427	0.84543	0.91186	0.96303	1.1385	1.5427	0.80794	0.64851	0.99136	1.0786	0.95114	0.93946	0.93946	1.7894	4
-0.56011	-0.58154	-0.62723	-0.66243	-0.78314	-1.0612	-0.55575	-0.44609	-0.68192	-0.74193	-0.65425	-0.64622	-0.64622	-1.2308	4
0.55275	0.5739	0.61899	0.65373	0.77285	1.0472	0.54845	0.44023	0.67296	0.73218	0.64566	0.63773	0.63773	1.2147	4
0.63435	0.65863	0.71037	0.75024	0.88695	1.2018	0.62941	0.50522	0.77231	0.84027	0.74098	0.73187	0.73187	1.394	4
-0.0057424	-0.0059621	-0.0064306	-0.0067915	-0.008029	-0.010879	-0.0056977	-0.0045734	-0.0069912	-0.0076064	-0.0067076	-0.0066252	-0.0066252	-0.012619	4
-1.2875	-1.3368	-1.4418	-1.5227	-1.8002	-2.4392	-1.2775	-1.0254	-1.5675	-1.7054	-1.5039	-1.4854	-1.4854	-2.8292	4
0.57308	0.59501	0.64175	0.67777	0.80128	1.0857	0.56862	0.45642	0.69771	0.7591	0.6694	0.66118	0.66118	1.2593	4
0.85877	0.89163	0.96168	1.0157	1.2007	1.627	0.85209	0.68395	1.0455	1.1375	1.0031	0.99079	0.99079	1.8871	4
0.20848	0.21646	0.23346	0.24657	0.2915	0.39498	0.20686	0.16604	0.25382	0.27615	0.24352	0.24053	0.24053	0.45813	4
0.88637	0.9203	0.9926	1.0483	1.2393	1.6793	0.87948	0.70594	1.0791	1.1741	1.0354	1.0226	1.0226	1.9478	4
0.1834	0.19042	0.20538	0.2169	0.25643	0.34746	0.18197	0.14606	0.22328	0.24293	0.21423	0.21159	0.21159	0.40302	4
-0.43373	-0.45033	-0.48571	-0.51296	-0.60644	-0.82173	-0.43035	-0.34543	-0.52805	-0.57452	-0.50663	-0.50041	-0.50041	-0.95311	4
-0.13676	-0.19559	-0.21096	-0.2228	-0.23567	-0.30055	-0.17913	-0.17249	-0.19749	-0.16635	-0.21234	-0.19776	-0.21734	-0.27129	5
-0.16571	-0.237	-0.25561	-0.26996	-0.28556	-0.36417	-0.21705	-0.209	-0.2393	-0.20157	-0.2573	-0.23963	-0.26335	-0.32872	5
0.49989	0.71495	0.77112	0.8144	0.86145	1.0986	0.65477	0.63051	0.72191	0.60808	0.77619	0.72289	0.79446	0.99164	5
0.11457	0.16386	0.17674	0.18666	0.19744	0.2518	0.15007	0.14451	0.16546	0.13937	0.1779	0.16568	0.18209	0.22728	5
-0.42655	-0.61005	-0.65798	-0.69491	-0.73506	-0.93742	-0.5587	-0.538	-0.61599	-0.51886	-0.66231	-0.61682	-0.67789	-0.84615	5
0.41729	0.59681	0.6437	0.67982	0.7191	0.91707	0.54657	0.52632	0.60262	0.5076	0.64793	0.60343	0.66318	0.82778	5
0.30373	0.43439	0.46852	0.49482	0.52341	0.6675	0.39783	0.38309	0.43862	0.36946	0.4716	0.43922	0.4827	0.60251	5
-0.10472	-0.14977	-0.16154	-0.17061	-0.18046	-0.23015	-0.13717	-0.13208	-0.15123	-0.12739	-0.1626	-0.15144	-0.16643	-0.20774	5
-1.027	-1.4688	-1.5842	-1.6731	-1.7697	-2.2569	-1.3451	-1.2953	-1.4831	-1.2492	-1.5946	-1.4851	-1.6321	-2.0372	5
-0.075619	-0.10815	-0.11665	-0.12319	-0.13031	-0.16619	-0.099047	-0.095377	-0.1092	-0.091985	-0.11741	-0.10935	-0.12018	-0.15001	5
-0.35049	-0.50128	-0.54066	-0.571	-0.60399	-0.77028	-0.45908	-0.44207	-0.50616	-0.42635	-0.54422	-0.50684	-0.55702	-0.69528	5
Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class

Price	City MPG	High Way MPG	# Cylinders	Engine Size	Horse Power	RPM at max HP	Revs/min	Fuel Tank Capacity	Passenger Capacity	Length	Width	Wheel Base	Weight	Class
0.39298	0.56203	0.60619	0.64021	0.6772	0.86364	0.51473	0.49565	0.56751	0.47803	0.61018	0.56827	0.62454	0.77955	5
0.47013	0.67237	0.7252	0.7659	0.81015	1.0332	0.61578	0.59296	0.67892	0.57187	0.72997	0.67984	0.74715	0.93259	5
-0.18726	-0.26782	-0.28886	-0.30508	-0.3227	-0.41154	-0.24528	-0.23619	-0.27043	-0.22779	-0.29076	-0.2708	-0.29761	-0.37147	5
0.03889	0.05562	0.05999	0.063357	0.067018	0.085468	0.050939	0.049051	0.056162	0.047307	0.060385	0.056238	0.061806	0.077146	5
-0.82285	-1.1768	-1.2693	-1.3405	-1.418	-1.8084	-1.0778	-1.0379	-1.1883	-1.0009	-1.2777	-1.1899	-1.3077	-1.6323	5
-0.22502	-0.32182	-0.34711	-0.36659	-0.38777	-0.49452	-0.29473	-0.28381	-0.32496	-0.27372	-0.34939	-0.3254	-0.35761	-0.44637	5
-0.25499	-0.36469	-0.39334	-0.41542	-0.43942	-0.56039	-0.33399	-0.32162	-0.36824	-0.31018	-0.39593	-0.36874	-0.40525	-0.50583	5
0.62662	0.8962	0.96661	1.0209	1.0798	1.3771	0.82076	0.79035	0.90493	0.76224	0.97297	0.90615	0.99586	1.243	5
0.45012	0.64377	0.69435	0.73332	0.77568	0.98924	0.58958	0.56774	0.65004	0.54754	0.69891	0.65092	0.71536	0.89292	5
0.35757	0.5114	0.55157	0.58253	0.61619	0.78583	0.46835	0.451	0.51638	0.43496	0.5552	0.51707	0.56827	0.70931	5
0.54405	0.7781	0.83924	0.88634	0.93755	1.1957	0.71261	0.68621	0.78568	0.6618	0.84476	0.78674	0.86464	1.0792	5
0.098869	0.1414	0.15251	0.16107	0.17038	0.21728	0.1295	0.1247	0.14278	0.12027	0.15352	0.14297	0.15713	0.19613	5
0.56851	0.81308	0.87697	0.92618	0.9797	1.2494	0.74465	0.71705	0.821	0.69155	0.88273	0.82211	0.90351	1.1278	5
-0.1524	-0.21797	-0.23509	-0.24829	-0.26263	-0.33494	-0.19962	-0.19223	-0.22009	-0.18539	-0.23664	-0.22039	-0.24221	-0.30233	5
0.16086	0.23006	0.24813	0.26206	0.2772	0.35352	0.2107	0.20289	0.2323	0.19567	0.24977	0.23261	0.25564	0.3191	5
0.18947	0.27099	0.29228	0.30868	0.32652	0.41641	0.24818	0.23898	0.27363	0.23048	0.2942	0.274	0.30112	0.37586	5
-0.019071	-0.027276	-0.029419	-0.03107	-0.032865	-0.041913	-0.02498	-0.024054	-0.027541	-0.023199	-0.029612	-0.027579	-0.030309	-0.037832	5
0.31299	0.44764	0.48281	0.5099	0.53936	0.68785	0.40996	0.39477	0.452	0.38073	0.48598	0.45261	0.49742	0.62088	5
-1.046	-1.496	-1.6135	-1.704	-1.8025	-2.2987	-1.37	-1.3193	-1.5105	-1.2724	-1.6241	-1.5126	-1.6623	-2.0749	5
0.25676	0.36722	0.39607	0.4183	0.44247	0.56429	0.33631	0.32385	0.3708	0.31233	0.39868	0.3713	0.40806	0.50934	5
0.31784	0.45457	0.49029	0.5178	0.54772	0.69851	0.41631	0.40089	0.459	0.38663	0.49351	0.45962	0.50513	0.6305	5
0.30552	0.43696	0.47129	0.49774	0.52649	0.67144	0.40018	0.38535	0.44121	0.37164	0.47438	0.44181	0.48555	0.60606	5
0.28661	0.40991	0.44212	0.46693	0.49391	0.62989	0.37541	0.3615	0.41391	0.34864	0.44503	0.41446	0.4555	0.56856	5
0.542	0.77517	0.83608	0.883	0.93402	1.1912	0.70993	0.68362	0.78272	0.65931	0.84157	0.78378	0.86138	1.0752	5
0.40138	0.57406	0.61916	0.65391	0.69169	0.88212	0.52574	0.50626	0.57965	0.48825	0.62323	0.58043	0.6379	0.79623	5
0.5861	0.83824	0.90409	0.95483	1.01	1.2881	0.76768	0.73924	0.8464	0.71294	0.91004	0.84754	0.93146	1.1626	5
0.2815	0.40261	0.43424	0.45861	0.48511	0.61866	0.36872	0.35506	0.40653	0.34243	0.43709	0.40708	0.44738	0.55842	5
-0.54078	-0.77342	-0.83418	-0.881	-0.9319	-1.1885	-0.70832	-0.68207	-0.78095	-0.65781	-0.83967	-0.782	-0.85943	-1.0727	5
-1.058	-1.5132	-1.632	-1.7236	-1.8232	-2.3252	-1.3858	-1.3345	-1.5279	-1.287	-1.6428	-1.53	-1.6814	-2.0988	5
0.2149	0.30735	0.33149	0.3501	0.37033	0.47228	0.28148	0.27105	0.31034	0.26141	0.33367	0.31076	0.34153	0.42629	5
0.55221	0.78978	0.85183	0.89963	0.95161	1.2136	0.7233	0.6965	0.79747	0.67173	0.85743	0.79855	0.87761	1.0954	5

-0.18212	-0.26048	-0.28094	-0.29671	-0.31385	-0.40025	-0.23855	-0.22971	-0.26301	-0.22154	-0.28279	-0.26337	-0.28944	-0.36128	5
-0.655	-0.93678	-1.0104	-1.0671	-1.1287	-1.4395	-0.85793	-0.82614	-0.94591	-0.79676	-1.017	-0.94718	-1.041	-1.2993	5
-0.98664	-1.4111	-1.522	-1.6074	-1.7002	-2.1683	-1.2923	-1.2444	-1.4248	-1.2002	-1.532	-1.4268	-1.568	-1.9572	5
0.46564	0.66597	0.71829	0.7586	0.80243	1.0233	0.60991	0.58731	0.67245	0.56642	0.72301	0.67336	0.74003	0.9237	5
0.0077756	0.011121	0.011994	0.012668	0.013399	0.017088	0.010185	0.0098072	0.011229	0.0094584	0.012073	0.011244	0.012357	0.015424	5
0.26942	0.38533	0.4156	0.43892	0.46428	0.5921	0.35289	0.33982	0.38908	0.32773	0.41833	0.3896	0.42818	0.53445	5
0.60123	0.85988	0.92744	0.97949	1.0361	1.3213	0.7875	0.75832	0.86825	0.73135	0.93353	0.86942	0.95551	1.1927	5
0.49818	0.7125	0.76848	0.81161	0.8585	1.0949	0.65253	0.62835	0.71944	0.606	0.77354	0.72041	0.79174	0.98825	5

Appendix I

MATLAB Code of the Practical Experiment for the Multi-Layer Perceptron

Code for the Training Process:

```
load CarsTrain.mat X ;

[n m] = size(X);
Samples=X(:,1:14);
Vlabels=X(:,15);

for i=1:n
  switch Vlabels(i)
      case 1
         Ydes(i,:)=[0 0 0 0 1];% Class 1
      case 2
         Ydes(i,:)=[0 0 0 1 0];% Class 2
      case 3
         Ydes(i,:)=[0 0 1 0 0];% Class 3
      case 4
         Ydes(i,:)=[0 1 0 0 0];% Class 4
      case 5
         Ydes(i,:)=[1 0 0 0 0];% Class 5
  end
end;

IR=minmax(Samples');
%break;
% Experiment 1
%net = newff(IR, [5],{'logsig'},'traingdx');

% Experiment 2
%net = newff(IR, [5 5],{'logsig' 'logsig'},'traingdx');

% Experiment 3
%net = newff(IR, [10 5],{'logsig' 'logsig'},'traingdx');

% Experiment 4
%net = newff(IR, [15 5],{'logsig' 'logsig'},'traingdx');

% Experiment 5
% net = newff(IR, [5 5 5],{'logsig' 'logsig' 'logsig'},'traingdx');

% Experiment 6
%net = newff(IR, [10 5 5],{'logsig' 'logsig' 'logsig'},'traingdx');
```

```
% Experiment 7
%net = newff(IR, [15 5 5],{'logsig' 'logsig' 'logsig'},'traingdx');

% Experiment 8
%net = newff(IR, [15 10 5],{'logsig' 'logsig' 'logsig'},'traingdx');

% Experiment 9
net = newff(IR, [15 15 5],{'logsig' 'logsig' 'logsig'},'traingdx');
net.trainParam.epochs = 2500;
net = train(net,Samples',Ydes');
Y = sim(net,Samples');
Compare=[Y' Ydes];
% Accuracy Computation
Er=0;
ConfMat = [0 0 0 0 0;0 0 0 0 0;0 0 0 0 0;0 0 0 0 0;0 0 0 0 0];
for j=1:n
   [mx ix]=max(Compare(j,1:5));

      if Compare(j,6:10) == [0 0 0 0 1]
        if(Compare(j,ix+5)~=1)
         Er=Er+1;
         if ix==1
            ConfMat (1,5) = ConfMat (1,5)+1;
         end;
          if ix==2
            ConfMat (1,4) = ConfMat (1,4)+1;
         end;
         if ix==3
            ConfMat (1,3) = ConfMat (1,3)+1;
         end;
         if ix==4
            ConfMat (1,2) = ConfMat (1,2)+1;
         end;
      else
              ConfMat (1,1) = ConfMat (1,1)+1;
         end ;
      end;

      if Compare(j,6:10) == [0 0 0 1 0]
        if(Compare(j,ix+5)~=1)
         Er=Er+1;
         if ix==1
            ConfMat (2,5) = ConfMat (2,5)+1;
         end;
          if ix==2
        ConfMat (2,4) = ConfMat (2,4)+1;
    end;
    if ix==3
        ConfMat (2,3) = ConfMat (2,3)+1;
    end;
    if ix==5
        ConfMat (2,1) = ConfMat (2,1)+1;
    end;
 else
           ConfMat (2,2) = ConfMat (2,2)+1;
    end ;
 end;

if Compare(j,6:10) == [0 0 1 0 0]
   if(Compare(j,ix+5)~=1)
    Er=Er+1;
    if ix==1
        ConfMat (3,5) = ConfMat (3,5)+1;
    end;
     if ix==2
        ConfMat (3,4) = ConfMat (3,4)+1;
    end;
    if ix==4
        ConfMat (3,2) = ConfMat (3,2)+1;
    end;
    if ix==5
        ConfMat (3,1) = ConfMat (3,1)+1;
    end;
 else
           ConfMat (3,3) = ConfMat (3,3)+1;
    end ;
 end;

if Compare(j,6:10) == [0 1 0 0 0]
   if(Compare(j,ix+5)~=1)
    Er=Er+1;
    if ix==1
        ConfMat (4,5) = ConfMat (4,5)+1;
    end;
     if ix==3
        ConfMat (4,3) = ConfMat (4,3)+1;
    end;
    if ix==4
        ConfMat (4,2) = ConfMat (4,2)+1;
    end;
    if ix==5
```

```
          ConfMat (4,1) = ConfMat (4,1)+1;
        end;
      else
            ConfMat (4,4) = ConfMat (4,4)+1;
        end ;
      end;

     if Compare(j,6:10) == [1 0 0 0 0]
       if(Compare(j,ix+5)~=1)
        Er=Er+1;
        if ix==2
           ConfMat (5,4) = ConfMat (5,4)+1;
        end;
         if ix==3
           ConfMat (5,3) = ConfMat (5,3)+1;
        end;
        if ix==4
           ConfMat (5,2) = ConfMat (5,2)+1;
        end;
        if ix==5
           ConfMat (5,1) = ConfMat (5,1)+1;
        end;
     else
            ConfMat (5,5) = ConfMat (5,5)+1;
     end ;
   end;

end;

Training_Accuracy = 100*((n-Er)/n);
```

<u>Code for the Testing Process:</u>

```
load CarsTest.mat X

[n m] = size(X);
Samples=X(:,1:14);
Vlabels=X(:,15);

for i=1:n
  switch Vlabels(i)
     case 1
        Ydes(i,:)=[0 0 0 0 1];% Class 1
     case 2
        Ydes(i,:)=[0 0 0 1 0];% Class 2
     case 3
```

```
          Ydes(i,:)=[0 0 1 0 0];% Class 3
       case 4
          Ydes(i,:)=[0 1 0 0 0];% Class 4
       case 5
          Ydes(i,:)=[1 0 0 0 0];% Class 5
    end
end;

IR=minmax(Samples');
Y = sim(net,Samples');
Compare=[Y' Ydes];

% Accuracy Computation
Er=0;
ConfMat_Test = [0 0 0 0 0;0 0 0 0 0;0 0 0 0 0;0 0 0 0 0;0 0 0 0 0];
for j=1:n
   k=0;
   [mx ix]=max(Compare(j,1:5));

       if Compare(j,6:10) == [0 0 0 0 1]
         if(Compare(j,ix+5)~=1)
          Er=Er+1;
          if ix==1
             ConfMat_Test (1,5) = ConfMat_Test (1,5)+1;
          end;
           if ix==2
             ConfMat_Test (1,4) = ConfMat_Test (1,4)+1;
          end;
          if ix==3
             ConfMat_Test (1,3) = ConfMat_Test (1,3)+1;
          end;
          if ix==4
             ConfMat_Test (1,2) = ConfMat_Test (1,2)+1;
          end;
       else
               ConfMat_Test (1,1) = ConfMat_Test (1,1)+1;
          end ;
       end;

     if Compare(j,6:10) == [0 0 0 1 0]
         if(Compare(j,ix+5)~=1)
          Er=Er+1;
          if ix==1
             ConfMat_Test (2,5) = ConfMat_Test (2,5)+1;
          end;
           if ix==2
           ConfMat_Test (2,4) = ConfMat_Test (2,4)+1;
       end;
       if ix==3
           ConfMat_Test (2,3) = ConfMat_Test (2,3)+1;
       end;
       if ix==5
           ConfMat_Test (2,1) = ConfMat_Test (2,1)+1;
       end;
    else
              ConfMat_Test (2,2) = ConfMat_Test (2,2)+1;
       end ;
    end;

if Compare(j,6:10) == [0 0 1 0 0]
      if(Compare(j,ix+5)~=1)
       Er=Er+1;
       if ix==1
           ConfMat_Test (3,5) = ConfMat_Test (3,5)+1;
       end;
        if ix==2
           ConfMat_Test (3,4) = ConfMat_Test (3,4)+1;
       end;
       if ix==4
           ConfMat_Test (3,2) = ConfMat_Test (3,2)+1;
       end;
       if ix==5
           ConfMat_Test (3,1) = ConfMat_Test (3,1)+1;
       end;
    else
              ConfMat_Test (3,3) = ConfMat_Test (3,3)+1;
       end ;
    end;
   if Compare(j,6:10) == [0 1 0 0 0]
      if(Compare(j,ix+5)~=1)
       Er=Er+1;
       if ix==1
           ConfMat_Test (4,5) = ConfMat_Test (4,5)+1;
       end;
        if ix==3
           ConfMat_Test (4,3) = ConfMat_Test (4,3)+1;
       end;
       if ix==4
           ConfMat_Test (4,2) = ConfMat_Test (4,2)+1;
       end;
       if ix==5
           ConfMat_Test (4,1) = ConfMat_Test (4,1)+1;
```

```
          end;
        else
                ConfMat_Test (4,4) = ConfMat_Test (4,4)+1;
          end ;
        end;

      if Compare(j,6:10) == [1 0 0 0 0]
          if(Compare(j,ix+5)~=1)
           Er=Er+1;
           if ix==2
              ConfMat_Test (5,4) = ConfMat_Test (5,4)+1;
           end;
            if ix==3
              ConfMat_Test (5,3) = ConfMat_Test (5,3)+1;
           end;
           if ix==4
              ConfMat_Test (5,2) = ConfMat_Test (5,2)+1;
           end;
           if ix==5
              ConfMat_Test (5,1) = ConfMat_Test (5,1)+1;
           end;
        else
                ConfMat_Test (5,5) = ConfMat_Test (5,5)+1;
        end ;
      end;
end;

Testing_Accuracy = 100*((n-Er)/n);
```

CURRICULUM VITAE

NAME:	Fadi Samih Omar Mehanna
ADDRESS:	11 El-Zahraa St. off Mosadak St. Dokki, Giza 12311 Egypt
DOB:	Dubai, UAE - June 16, 1976
Email:	fadi_mehanna@hotmail.com
EDUCATION & TRAINING:	B.A., German Language Ain Shams University, Cairo, Egypt 1993 - 1997