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

6.1. Introduction

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

6.1.1. Why Python

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

Python is a very powerful programming language and is used in a wide variety of application domains. In the area of machine learning it has proved to be very helpful and effective. One of the main reasons of using this language is its intuitive object orientation as OOP paradigm is the most commonly followed paradigm these days. It has full modularity and supports hierarchical
packages. Since our machine learning problems revolve around different types of datasets, we need to be careful about the data types supported by the programming language we use. Python has a very high level dynamic data types. It has a number of extensive standard libraries and third party modules for virtually every task. It can be easily embedded within applications as a scripting interface. More importantly, Python supports portability. We can run the same source code without changing across all implementations. It runs everywhere. It is available for Windows, Linux/Unix, OS/2, Mac, Amiga, and others.

6.1.2. Python Machine Learning tool

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

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

Without the use of such machine learning tools, we would have to write the entire code ourselves for all the machine learning tasks e.g. for carrying out cross validation for comparing the
machine learning algorithms, or for loading data and so on. Machine learning toolkits ease the programming by providing in built routines for these tasks thus providing flexibility in experimenting. All we need to do is access these routines from our code.

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

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

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

6.2. Combined Learners

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

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

6.2.1. Types of Combination Techniques

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

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

\[
y = \arg\max_{c_j \in C} \sum P(c_j \mid h_i) P(T \mid h_i) P(h_i), \ h_i \in H
\]
where \( y \) denotes the predicted class, \( C \) represents the set of all possible classes, \( H \) is the hypothesis space, \( P \) refers to a probability, and \( T \) is the training data.

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

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

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

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

If \( A \) is a learning algorithm and \( T \) is a set of training data, the process of bagging takes \( N \) samples \( S_1, \ldots, S_N \), from \( T \). The algorithm is then applied to each sample independently to make \( N \) models \( h_1, \ldots, h_N \). When a new query \( q \) has to be classified, these models are combined by a simple voting scheme, and the query is assigned a class that has been predicted most often among the \( N \) models. Figure 6.1 (http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf) shows the process of bagging diagrammatically. For generating training datasets, bagging uses bootstrap and the learners are trained using an unstable learning procedure, and an average is taken during testing [Breiman 1996]. This method works effectively if the base learner is unstable i.e. if it is highly sensitive to data i.e. small changes in the training set cause a large difference in the generated learner. This method can be used both for classification
and regression. In case of regression, instead average, median is taken at the time of testing.

![Figure 6.1: Bagging](http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf)

Boosting [Schapire, 1990] is a process which trains the new instances and combines the learners incrementally in a way such that the focus is laid on the training instances that were previously wrongly classified. In this method the learner is trained on the mistakes of the previous learners. Bagging is based on data variation through a learner’s instability and boosting is based on data variation through a learner’s weakness. A learner is said to be weak if it derives models that perform slightly better than random guessing. In a weak learner, the error probability is \( \frac{1}{2} \). It means for a two-class problem it is better than random guessing and a strong learner has small error probability. The most common example of boosting is adaptive boosting, AdaBoost [Freund and Schapire, 1996]. The process of boosting works by supposing that if a weak learner is run on different distributions repeatedly over the training data, and if the weak classifiers are combined into a single classifier, then it can be made stronger, as illustrated in Figure 6.2 (http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf). The main disadvantage of the boosting method is its need for large training data. AdaBoost does
not suffer this problem as it uses the same training set over and over and thus the training data need not be large, but the classifiers should be simple so that they do not overfit.

![Diagram of Boosting](http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf)

**Figure 6.2: Boosting**

- **Stacking**: This method exploits *variation among learners* in which several learning algorithms are applied to the same application’s data. This method is proposed by Wolpert in 1992. In this method a number of different learning algorithms are run against the dataset which creates a series of models. Then the actual dataset is modified by replacing its each instance by the values that each model predicts for that instance. This creates a new dataset which is given to a new learner that builds the model, as illustrated in Figure 6.3 ([http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf](http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf)). Whenever a new query instance q has to be classified it is first passed through all the learners which create a new query instance q’. Then the model takes it as an input and the final classification for q is produced. For better results it is important in stacked generalization that the learners should be as different as possible so that they will complement each other and these learners should be based on different learning algorithms.
• **Cascading:** This method also follows *variation among learners* approach like stacking but it differs from stacking because stacking uses the learners in parallel whereas cascading uses the learners in sequence. Cascading is a process having multiple stages in which learners are used in sequence i.e. the next learner is used only if the preceding ones are not confident [Alpaydin, 2010]. This method was proposed by Gama and Brazdil. Figure 6.4 ([http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf](http://dml.cs.byu.edu/~cgc/docs/mldm_tools/Reading/ModelCombination.pdf)) shows this process.

**Figure 6.3: Stacking**

In cascading the data from the base-level learners is not fed into a single meta-level learner. But each base-level learner also acts as a kind of meta-level learner for the
learner preceding it. The inputs that are fed to the learner consist of the inputs to learner preceding it together with the class probabilities produced by the model induced by the preceding learner. At each step only a single learner is used and the number of steps is unlimited. A new query instance $q$ is converted into a query instance $q'$ by gathering data through the steps of the cascade. The last model of the cascade produces the final classification.

### 6.2.2. Related Literature

A lot of research work has been carried out in this field. This section presents the work done in the direction of combined learners. A technique called attribute bagging has been developed for improving accuracy and stability of classifier ensembles induced using random subsets of features. This method has been compared with bagging and other methods on a hand-pose recognition dataset and has shown better results than bagging and other methods both in terms of accuracy and stability [Bryll et al., 2002]. Bagging was first introduced by Leo Breiman. He created a method called Bagging Predictors for generating multiple versions of a predictor and used these to create an aggregated predictor [Breiman, 1996]. A Bayesian version of bagging based on the Bayesian bootstrap has been developed. The Bayesian bootstrap has shown to resolve a theoretical problem with ordinary bagging and resulted in more efficient estimators [Clyde and Lee, 2000]. An experimental comparison has been carried out between bagging, boosting and randomization for improving the performance of the decision-tree algorithm C4.5. The experiments have shown that randomization is slightly superior to bagging but not as accurate as boosting in situations with little or no classification noise [Dietterich, 1999]. However, it has been shown that in noisy settings bagging performs much more robustly than boosting. A method of ensemble technique has been developed in which voting methodology of bagging and boosting ensembles has been used with 10 subclassifiers in each one. It has been compared with simple bagging and boosting ensembles with 25 sub-classifiers, and also with other well known combining methods, on standard benchmark datasets and it has been shown that the new is the most accurate [Kotsiantis and Pintelas, 2004]. An algorithm called RankBoost has been developed for combining preferences based on the boosting approach to machine learning. Theoretical results have been shown describing the algorithm’s behavior both on the training data, and on new test data not seen during training. Two experiments have been carried
out to assess the performance of RankBoost. In the first experiment, the algorithm has been used to combine different web search strategies, each of which is a query expansion for a given domain. The second experiment has been a collaborative-filtering task for making movie recommendations [Freund et al., 2003]. A statistical perspective on boosting has been proposed with special emphasis on estimating potentially complex parametric or nonparametric models, including generalized linear and additive models as well as regression models for survival analysis. The practical aspects of boosting procedures for fitting statistical models have been illustrated by means of the dedicated open-source software package mboost [Buhlmann and Hothorn, 2007]. Theoretical and practical aspects of boosting and ensemble learning have been discussed and the helpful association that exists between boosting and the theory of optimization has been identified for easing the understanding of boosting [Meir and Ratsch, 2003]. Voting classification algorithms like bagging, boosting and variants have been compared in order to find which of these algorithms use perturbation, reweighting, and combination techniques, and which of the algorithms affect classification error. The authors have shown bias and variance decomposition of the error for showing bias and variance decomposition are influenced by different methods. This comparison has shown that bagging reduces variance of unstable methods, while boosting methods (AdaBoost and Arc-x4) reduce both the bias and variance of unstable methods but increase the variance for Naive-Bayes. It has been found that when probabilistic estimates are used along with no-pruning, then bagging shows an improvement. Mean-squared error of voting methods has been compared to non-voting methods and it has shown that the voting methods show reduction in the errors. They have also examined the problems that arise when boosting algorithms are practically implemented [Bauer and Kohavi, 1998]. Simple online bagging and boosting algorithms have been developed that perform as well as their batch counterparts. Lossless online algorithms for decision trees and Naïve Bayes models have been used [Oza and Russell, 2005]. Cohen has developed stacked sequential learning which is a sequential learning scheme in which an arbitrary base learner is improved so that it becomes aware of the labels of nearby examples. This method has been assessed on various problems. It has been shown that on these problems, the performance of non-sequential base learners improves by sequential stacking; that the performance of learners specially designed for sequential tasks is improved by sequential stacking [Cohen, 2005]. A learning method using multiple stacking for named entity recognition has been proposed which employs stacked
learners using the tags predicted by the lower level learners. This approach has been applied to the CoNLL-2002 shared task to improve a base system [Tsukamoto et al., 2002]. Different methods for interpreting the results of multiple, cascading machine learners have been explored. Each of these methods perform a different task. A framework for modeling cascading learners as a directed acyclic graph has been developed, which has allowed a construction of three-way contingency tables on which various independence tests has been performed. These independence tests have provided insight into how the various learners’ performance depends on their predecessor in the chain and/or the inputs themselves [Michelson and Macskassy, 2010]. A technique of localized cascade generalization of weak classifiers has been developed. Using this technique some local regions have been pointed out that have like properties and the cascade generalization of local experts has been used for explaining the relationship between the data characteristics and the target class. This technique has been compared with other well known combining methods using weak classifiers as base-learners, on standard benchmark datasets and it has been shown that this technique is more accurate [Kotsiantis, 2008]. A method has been proposed based on the enrichment of a set of independent labeled datasets by the results of clustering, and a supervised method has been used to evaluate the interest of adding such new information to the datasets. The cascade generalization paradigm has been adapted in the case where an unsupervised and a supervised learner are combined [Candillier et al., 2006]. Bagging, stacking, boosting and error correcting output codes are the main four methods of combining multiple models. These have been discussed covering seven methods of combining multiple learners i.e., voting, bagging, cascading, error-correcting output codes, boosting, mixtures of experts, and stacked generalization [Witten and Frank, 2000]. A theoretical framework for combining classifiers in the two main fusion scenarios has been developed. These two main fusion scenarios are fusion of opinions based on identical and on distinct representations [Kittler, 1998]. For the first scenario i.e. the shared representation they showed that here fusion has been performed with the aim of obtaining a better estimation of the appropriate a posteriori class probabilities. For the second scenario i.e. for the distinct representations it has been pointed out that the techniques based on the benevolent sum-rule fusion are more flexible to errors than those derived from the severe product rule.
6.3. Our approach towards combining learners

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

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

Consider we have to combine N learners (l₁, l₂, … lₙ). We represent each learner by lᵢ and the prediction of each learner lᵢ by dᵢ(x). If y represents the final prediction, we can calculate y from the individual predictions of learners, i.e.

\[ y = f(d₁, d₂, \ldots, dₙ \mid Φ) \]

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

Choose Cᵢ if yᵢ = max yₖ where k = 1 to K

From equation 1 we get,

\[ yᵢ = Σ wⱼdⱼᵢ \text{ where } wⱼ ≥ 0, Σ wⱼ = 1 \]
In case of classification, the weights approximate to the learner probabilities. Therefore,

\[ w_j = P(l_j) \]

\[ d_{ji} = P(C_i \mid x, l_j) \]

The above equation can be rewritten as

\[ P(C_i \mid x) = \sum P(C_i \mid x, l_j) P(l_j) \text{ for all learners } l_j \]

The class probabilities are calculated using this formula.

**6.3.1. Procedure of our approach**

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

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

- For each learner \( l_j \) (\( j = 1 \text{ to } N \)) in the combination
  create the classifier \( m_j \) for \( l_j \) by training on the dataset \( D \)
  \[ m_j = l_j(D) \]

- For each class \( C_i \) (\( i = 1 \text{ to } K \)) in the data
  For each classifier \( m_j \) (\( j = 1 \text{ to } N \)
Calculate $P(C_i) = P(C_i \mid x, m_j)$ that represents the probability that the classifier $m_j$ assigns to the class $C_i$.

- Finally, we choose the class with the highest predicted probability, or the class with maximum value for $P(C_i)$ i.e.
  Choose $C_i$ if $P(C_i)$ has the maximum value among all $P(C_i)$’s

![Diagram](image.png)

**Figure 6.5: Flow of the combined technique**

### 6.3.2. Experimental Setup

As mentioned earlier, for the implementation of the above discussed procedure, we used Python programming and for applying machine learning methods we used Python machine learning tool called Orange. We implemented this approach on the classification problem used in the previous chapter. The dataset ([http://www.hakank.org/weka/credit.arff](http://www.hakank.org/weka/credit.arff)) that we used for our experiment for implementing our procedure is the Australian Credit Approval dataset from UCI Repository of Machine Learning Databases and Domain theories ([http://archive.ics.uci.edu/ml/datasets.html](http://archive.ics.uci.edu/ml/datasets.html)). It is the same dataset that we used in previous chapter for the evaluation of various machine learning algorithms, and its description has already been provided so we skip it here. However, for using the dataset in Orange we had to change its format from ARFF (supported in WEKA) to tab delimited format supported in Orange. The dataset is split into the training and the test sets as done in the previous chapter i.e.
“trainingcredit” and “testingcredit”. The main reason for using the same dataset is to compare the accuracy of the individual learners used in the previous chapter with the accuracy of the combined approach. As discussed earlier, Orange provides a number of inbuilt routines for performing various machine learning tasks. Without its use, we would have to write the entire code ourselves for all the machine learning tasks e.g. for carrying out cross validation for comparing the machine learning algorithms, or for loading data and so on. We provide a list of routines that we used for our approach of combining various learners:

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

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

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

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

- Finally, for evaluation of our learners, we use cross validation method just as we used in the previous chapter.
  
  ```python
  Evaluationresult = orange.crossValidation(learners, D)
  ```
The experiment was carried out in Python machine learning tool. For our experiment we used three learners for combination, i.e. we kept $N = 3$. The algorithms that we used are RandomForest, Naivebayes, and kNN. Then we performed cross validation with 10 folds just like in previous chapter. We split our dataset into training and testing sets.

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

![Figure 6.6: Running a Script in Interactive Window in Python](image)
6.3.3. Results

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

Accuracy = \[
\frac{tp + tn}{tp + fp + tn + fn}
\]

Precision(P) = \[
\frac{tp}{tp + fp}
\]

Recall(R) = \[
\frac{tp}{tp + fn}
\]

F-Measure = \[
\frac{2 \cdot P \cdot R}{P + R}
\]

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

<table>
<thead>
<tr>
<th>LEARNERS</th>
<th>ACCURACY</th>
<th>BRIER SCORE</th>
<th>F-MEASURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomForest</td>
<td>0.845</td>
<td>0.217</td>
<td>0.861</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>0.864</td>
<td>0.236</td>
<td>0.881</td>
</tr>
<tr>
<td>kNN</td>
<td>0.831</td>
<td>0.247</td>
<td>0.848</td>
</tr>
<tr>
<td>Combinedlearner</td>
<td>0.870</td>
<td>0.219</td>
<td>0.885</td>
</tr>
</tbody>
</table>

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

6.4. Feature Space Design

As discussed in Subsection 3.1.3 of Chapter 3, data preprocessing [Zhang et al., 2002] is an important task of machine learning. Initially the data collected is not directly suitable for training and therefore requires some processing before it can be used for example it may have missing feature values or noise. A number of pre-processing methods have been developed and the decision of deciding which one to use varies according to the situations. If the collected data contains some missing features then a method for handling missing data [Batista & Monard, 2003] is used. Similarly, there are methods for detecting and handling noise [Hodge & Austin, 2004]. Some of the problems with the collected real world data are: data can be incomplete i.e.
some attribute values may be missing, or it may lack certain important attributes, or it may consist of only aggregate data; there can be presence of noise i.e. it may contain errors or outliers; the data may be inconsistent i.e. containing variations in codes or names. Data preprocessing is performed in order to prepare the data for input into machine learning and mining processes. This involves transforming the data for improving its quality and hence the performance of the machine learning algorithms, such as predictive accuracy and reducing the learning time. Once the data preprocessing is complete we get a final training set. A well-known algorithm has been presented for each step of data pre-processing [Kotsiantis et al., 2006].

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

As we have shown before, for our machine learning tasks, data is represented as a table of examples or instances. It is called the dataset. Every instance in the dataset has a fixed number of attributes, or features, along with a label that denotes its class. The features of a dataset contain the information about the problem that we are dealing with and help in the classification process. Usually we believe that if the number of features or attributes is increased in the dataset, it will increase the efficiency of classification. However, by increasing the features there are chances of degradation of the classifier performance [Bishop, 1995]. Usually in many real-world problems, there are a large number of features in the dataset, most of which are irrelevant or redundant. Therefore, an important task in machine learning is deciding and choosing which of the features are relevant and which are irrelevant. Before a classifier can move beyond the training data to make predictions about novel test cases, it must decide which features to use in these predictions.
and which to ignore. Therefore it is necessary to find subsets of the feature population that are relevant to the target class and worthy of focused analysis [Blum and Langley, 1997]. This process in which some of the features of the training set are selected and used for classification is called feature selection.

6.4.1. Feature Selection

The most important purpose of feature selection is to make a classifier more efficient by decreasing the size of the dataset. This is necessary for the classifiers that are costly to train e.g. NaiveBayes. The processing time and the cost of the classification systems are increased while their accuracy is decreased if irrelevant and additional features are used in the datasets used for classification. Therefore, it is very important to develop the techniques for selecting smaller feature subsets. However, we have to make sure that the subset which is selected is not so small that the accuracy rates are reduced and the results lack understandability. So it is very important that techniques must be developed that help to find an optimal subset of features from the superset of original features [Witten and Frank, 2000]. There are two ways in which feature selection can be carried out. These are the filter and wrapper approach [Liu and Motoda, 1998]. The filter approach selects a subset of the features that preserves as much as possible the relevant information found in the entire set of features [Kohavi and John, 1997; Freitas, 2002]. Some of the methods that implement filter approach are discussed here. The FOCUS algorithm [Almuallim and Dietterich, 1991] has been designed for noise-free Boolean domains and it follows the MIN-FEATURES bias. It examines all feature subsets and selects the minimal subset of features that is sufficient to predict the class targets for all records in the training set. Another feature selection method that has been developed is called Relief [Kira and Rendell, 1992]. It is an instance-based feature selection method. Relief-F is an extended version of Relief that has been developed for multi-class datasets whereas Relief was designed for two-class problems. In this method an instance is randomly sampled from the data and its nearest neighbor is located from the same and opposite class. The sampled instance is compared to the values of the features of the nearest neighbors and relevance scores for each feature are updated. The process is then carried out repeatedly for many instances. The main idea is that an attribute should be able to differentiate between instances from different classes and should have the same value for instances from the same class. Information gain and gain ratio [Quinlan, 1993] are good examples of measuring the relevance of features for decision tree induction. They use the
entropy measure to rank the features based on the information gained; the higher the gain the better the feature. Moore and Lee [Moore and Lee, 1994] proposed another model using an instance-based algorithm, called RACE, as the induction engine, and leave-one-out cross-validation (LOOCV) as the subset evaluation function. Searching for feature subsets is done using backward and forward hill-climbing techniques. John et al. [John et al., 1994] proposed a similar method and applied it to ID3 and C4.5 on real world domains. Langley et al. [Langley and Sage, 1994] also used LOOCV in a nearest-neighbor algorithm. Caruana et al. [Caruana and Freitag, 1994] test the forward and backward stepwise methods on the Calendar Apprentice domain, using the wrapper model and a variant of ID3 as the induction engine. Wrapper models are usually slower than filter models in the sense that inductive learning is carried out more than once.

6.4.2. Basic Steps in Feature Selection

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

- Initialize the learner.
  
  learner = Learner()

- Load the dataset in D,

  
  D = orange.Exampletable("trainingcredit")

  This loads the dataset that we have used i.e. Credit dataset in D.

- For creating the classifiers by training the learner on the dataset,

  
  Classifier = learner(D)

- Compute the relevance (R) of the features/attributes. This is done by applying the attribute measure method on the dataset (i.e. attMeasure(D)).

- Set some margin, say m, and remove all those features/attributes for which R < m, i.e. whose relevance is below the selected margin. This is done by applying a filter method on the dataset. Only the attributes having R > m are used for classification.

- Finally, use the learner on both the datasets and compare the accuracy.
6.4.3. Experiment and Results

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

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

<table>
<thead>
<tr>
<th>Learners</th>
<th>Accuracy Before feature selection</th>
<th>Accuracy After feature selection</th>
<th>F-Measure Before feature selection</th>
<th>F-Measure After feature selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomForest</td>
<td>0.845</td>
<td>0.852</td>
<td>0.861</td>
<td>0.867</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>0.864</td>
<td>0.864</td>
<td>0.881</td>
<td>0.880</td>
</tr>
<tr>
<td>kNN</td>
<td>0.831</td>
<td>0.825</td>
<td>0.848</td>
<td>0.845</td>
</tr>
<tr>
<td>CombinedLearner</td>
<td>0.870</td>
<td>0.868</td>
<td>0.885</td>
<td>0.879</td>
</tr>
</tbody>
</table>

Figure 6.11 shows the results of feature subset selection taking margin 0.020. Table 6.3 shows the comparison of the performances of the learners based on accuracy and F-Measure with and without feature selection for margin 0.020. It shows a decrease in the accuracy and F-Measure of
all the learners. After subset selection, only 6 attributes are chosen for classification and remaining attributes are ignored as there relevance is below the margin. But this decreases the overall accuracy of the learners. Hence, for our problem the optimal subset of features is obtained by keeping margin equal to 0.010, which corresponds to 11 out of 15 attributes.

![Table and Figure]

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

In Table 6.4 we have shown the comparison of learners on the basis of their F-Measures without feature selection and with feature selection at two different margins. It is clear that feature selection is important but only as long as it does not decrease the efficiency of the learners by discarding too many attributes on the basis of their relevance. At margin 0.010, learners perform better than without any margin. They show increased or similar efficiency depicting the fact that rest of the attributes were irrelevant. However, at margin 0.020, learners show decrease in performance indicating that too many attributes are being discarded and hence the chosen subset is not an optimal subset.
### Table 6.3: Before and after feature selection comparison of learners with margin 0.020

<table>
<thead>
<tr>
<th>Learners</th>
<th>Accuracy Before feature selection</th>
<th>Accuracy After feature selection</th>
<th>F-Measure Before feature selection</th>
<th>F-Measure After feature selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomForest</td>
<td>0.845</td>
<td>0.838</td>
<td>0.861</td>
<td>0.854</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>0.864</td>
<td>0.858</td>
<td>0.881</td>
<td>0.874</td>
</tr>
<tr>
<td>kNN</td>
<td>0.831</td>
<td>0.831</td>
<td>0.848</td>
<td>0.843</td>
</tr>
<tr>
<td>CombinedLearner</td>
<td>0.870</td>
<td>0.852</td>
<td>0.885</td>
<td>0.869</td>
</tr>
</tbody>
</table>

### Table 6.4: Comparing F-Measure at different margins

<table>
<thead>
<tr>
<th>Learners</th>
<th>F-Measure Before feature selection</th>
<th>F-Measure After feature selection (margin 0.010)</th>
<th>F-Measure After feature selection (margin 0.020)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RandomForest</td>
<td>0.861</td>
<td>0.867</td>
<td>0.854</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>0.881</td>
<td>0.880</td>
<td>0.874</td>
</tr>
<tr>
<td>kNN</td>
<td>0.848</td>
<td>0.845</td>
<td>0.843</td>
</tr>
<tr>
<td>CombinedLearner</td>
<td>0.885</td>
<td>0.879</td>
<td>0.869</td>
</tr>
</tbody>
</table>