AN INTERACTIVE WEB-BASED MACHINE-LEARNING SUITE

by

Pınar Yanardağ

B.S., Computer Engineering, Çanakkale Onsekiz Mart University, 2007

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2010

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisor Prof. Ethem Alpaydın for his invaluable supervision, academic feedback, support and endless patience during this thesis.

I would like to thank Prof. Fikret Gürgen and Assist. Prof. Olcay Taner Yıldız for being in my thesis Jury.

I would especially like to present my gratitude to Mehmet Gönen for counseling me in any matter where I get confused.

I would like to thank my collegues at TUBITAK UEKAE for encouraging me to pursue an academic career and members of Perceptual Intelligence Laboratory for providing such an amusing environment.

I would like to thank Google for supporting my academic studies since my undergraduate education with various scholarships as well as playing an important role in shaping my scientific perception.

I would like to thank stackoverflow.com family who taught me to discuss the discussion itself and helped me to solve challenging programming issues in my thesis.

Finally, I would like to thank my family for everything they have done for me.

ABSTRACT

AN INTERACTIVE WEB-BASED MACHINE-LEARNING SUITE

We propose a comprehensive and interactive web-based machine learning suite that will allow researchers and practitioners to use a wide collection of basic and experimental learning algorithms and sophisticated visualization and analysis tools. Component based frameworks incorporating data input/output, pre-processing, classification, clustering, regression and visualization schemes have been implemented before in various programming languages, for use on different platforms, to operate using a variety of data formats. ML-Lab includes a large variety of machine learning algorithms for resampling, feature selection and extraction, classification and ensemble methods, as well as tools to visualize the experimental results of statistical comparison and testing. It provides a sophisticated and easy-to-use interface for creating workflows and a component-based framework intended for both experienced users and also those who are just entering the field. The collection of machine learning algorithms are implemented in Python, a modern easy-to-use scripting language with clear but powerful syntax and extensive set of additional libraries. ML-Lab has an extensible architecture and allows adding new capabilities to the system infrastructure easily.

ÖZET

WEB TABANLI ETKİLEŞİMLİ YAPAY ÖĞRENME GERÇEKLEMESİ

Bu çalışmada, araştırmacılara ve kullanıcılara hem temel, hem de deneysel yapay öğrenme algoritmaları içeren web tabanlı, platform bağımsız ve kapsamlı bir görselleştirme ve analiz aracı sunuyoruz. Sınıflama, kümeleme, gerileme, ön-işleme ve görselleştirme algoritmalarını bir arada sunan bileşen tabanlı uygulamalar daha önce bir çok programlama dilinde, değişik platformlar üzerinde çalışacak şekilde ve değişik veri formatları üzerinde gerçekleştirildi. Ancak platform bağımlı bu uygulamalar, kullanıcılara değişik yapay öğrenme algoritmalarını hızlı ve kolay bir şekilde deneme ve karşılaştırma imkanı vermemektedir. ML-Lab örnekleme, özellik seçme ve çıkarma, sınıflama, görsel olarak karşılaştırma ve istatistiksel testler gibi bir çok metoda imkan vermektedir. ML-Lab kullanıcılara sunduğu geniş algoritma seçeneklerine ek olarak, kullanıcı dostu ve şık bir arayüz sunmaktadır. ML-Lab'ın bileşen tabanlı uygulama çatısı hem deneyimli kullanıcıları ve araştırmacıları, hem de yapay öğrenme alanına yeni adım atmış kullanıcıları hedeflemektedir. Bu çalışmadaki yapay öğrenme algoritmaları kullanımı kolay ve güçlü bir sözdizimine sahip olan Python betikleme dilinde gerçeklenmiştir. ML-Lab sisteme kolayca yeni algoritmalar ve eklentiler yaplabilmesi için genişletilebilir bir mimaride tasarlanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AUC	Area Under the Curve
AUPC	Area Under the PR Curve
FFS	Forward Feature Selection
FN	False Negative
FP	False Positive
k-NN	k-Nearest Neighbor Classifier
LDA	Linear Discriminant Analysis
PCA	Principle Component Analysis
PR	Precision Recall
ROC	Receiver Operating Characteristic
TN	True Negative
TP	True Positive

1. INTRODUCTION

"Find a bug in a program, and fix it, and the program will work today. Show the program how to find and fix a bug, and the program will work forever." - Oliver G. Selfridge [1].

Ever since computers were invented, researchers have been searching for ways to get computers to program themselves. Machine learning is the discipline that focuses on the design and development of algorithms that allow computers to make intelligent decisions, extract important relationships and recognize complex patterns, based on experience.

In general, any computer program that can improve its performance at some task through experience (or training) can be called a learning program [2]. Mitchell [2] defines learning as follows:

Definition: A computer program is said to **learn** from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**.

There are three components of a learning method: the concept description language, the learning element, and the performance element. The concept description language is used to represent a model (e.g. rules, probabilities, equations). The learning element uses the training examples to produce a model in description language. The performance element is used to make a prediction about a given observation.

Machine learning is part of artificial intelligence, but it is related to many different academic disciplines, especially probability theory and statistics, mathematics, pattern recognition and theoretical computer science.

Machine learning has a broad range of applications such as bioinformatics, computer vision, face detection, spam filtering, medical diagnosis, fraud detection, speech recognition and customer segmentation.

1.1. Basic Concepts and Terms

Before going on, some terminology used widely in Machine learning applications is defined below:

A *class* contains similar objects while objects from different classes are not similar. Some classes correspond to labels for different populations, for example whether a person is a male or female. In this case, we certainly know that males and females from separate classes. A class is represented as a random variable, for example: the customer will buy the product (class =1) or not (class = 0).

We can assume that there are n possible classes in the problem, labeled c_1 to c_n, organized as a set of labels $C = \{c_1, ..., c_n\}$ and that each object belongs to one and only one class [3].

A *data set* contains labeled information, for example a set of handwritten digits which are labeled into 10 classes.

A *feature* is a specification of an attribute and its value. It can be a numeric or categorical quantity used as input to a classifier (e.g.: "the weight of an person").

A *feature vector* is a vector of features, denoted by an x. In general, a classification function is a function defined on feature vectors and taking values in a set of class labels.

A *model* is a structure that summarizes a set of data, generally for prediction.

A *training set* is a collection $(X'_i, Y_i), ..., (X'_n, Y_n)$ where (X'_i, Y_i) are labeled examples used for training.

A *test set* is a collection $(X'_i, Y_i), ..., (X'_m, Y_m)$ that is used to validate the performance of a classifier.

1.2. Related Work

Before introducing our work, we discuss previous work. There is no web-based machine learning implementation yet, so we discuss platform-dependent solutions instead. Weka [4] and Orange [5] are the most commonly used frameworks.

Weka is a workbench that contains a collection of visualization tools and algorithms for data analysis and predictive modelling initially proposed in 1993 by the University of Waikato in New Zealand [4]. It supports data preprocessing, clustering, classification, regression, visualization, and feature selection methods.

Orange is another component based software suite and includes a range of preprocessing, modelling and data exploration techniques. Orange project is started at the University of Ljubljana, Slovenia in 1997 [5].

In the following tables, we compare ML-Lab with Weka and Orange. Table 1.1 shows that ML-Lab, Orange and Weka support the same classification algorithms, but ML-Lab does not include any clustering/regression method (yet).

Table 1.2 shows the comparison of evaluation methods. ML-Lab supports PR Curve while none of the other frameworks support it.

Table 1.1. Comparison of Algorithms

Algorithms	ML-Lab	Weka	Orange
k-NN	√	√	√
Naive Bayes	√	√	√
SVM	√	√	√
C4.5	√	√	√
Clustering	X	√	√
Regression	X	√	√

Table 1.2. Comparison of Evaluation Methods

Method	ML-Lab	Weka	Orange
ROC Curve	√	√	√
PR Curve	√	X	X
Cross Validation	√	√	√
Hypothesis Testing	√	√	X

Table 1.3 shows the comparison of dimensionality reduction methods of the three frameworks. ML-Lab supports LDA and ISOMAP, while others do not.

Table 1.3. Comparison of Dimensionality Reduction Methods

Method	ML-Lab	Weka	Orange
PCA	√	√	√
LDA	√	X	X
ISOMAP	√	X	X
Forward Feature Selection	√	√	√

In Table 1.4 we compare the technology (dependency, language, architecture, license) of the frameworks. ML-Lab is the only framework that is platform-independent while other frameworks need to be properly installed and configured for the platform.

Table 1.4. Comparison of Technology

Feature	ML-Lab	Weka	Orange
Programming Language	Python	C++ and Python	Java
Platform Depended	√	X	X
Extensible Architecture	√	√	√
License	GPL	GPL	GPL

1.3. Motivation

Component-based machine learning frameworks have become a popular application area in recent years. Weka [4] and Orange [5] support various machine learning techniques for pre-processing, classification and visualization but they don't provide a platform-independent solution.

Our motivation is to develop a platform-independent and web-based machine learning framework with an extensible architecture to which in the future different users, researchers and developers can contribute.

1.4. Outline of the Thesis

The outline of this thesis is as follows. In Chapter 2, we discuss the implemented machine learning and dimensionality reduction algorithms, resampling and evaluation methods. In Chapter 3, we discuss our solutions for a platform-independent machine learning framework, architecture, web interface and backend libraries. In Chapter 4, we describe the datasets we have used in the experiments, our experimental methodology, evaluation metrics and the results we have obtained. We conclude and outline future directions in Chapter 5.

2. METHODS

In this chapter, we discuss different methods for classification, dimensionality reduction, resampling and evaluation of the results that are implemented in ML-Lab.

2.1. Classification

The goal of classification is to build a set of models that can correctly predict the class of the different objects.

ML-Lab supports k-NN, Naive Bayes, SVM and C4.5 classification algorithms with a wide range of attributes.

2.1.1. k-NN Classification Algorithm

k-Nearest Neighbor (k-NN) algorithm is a method for classifying objects based on the closest training examples in the feature space. The k-NN classifier commonly use the Euclidean distance between a test sample and the specified training samples.

Assume that x_i is an input sample with d features $(\mathrm{x}_{i1}, \mathrm{x}_{i2}, \ldots, \mathrm{x}_{id})$ and assume n is the total number of input samples $(i = 1, 2, \ldots, n)$, the Euclidean distance between sample x_i and x_j $(j = 1, 2, \ldots, n)$ is defined as:

$$\|\mathrm{x}_i - \mathrm{x}_j\| = \sqrt{(\mathrm{x}_{i1} - \mathrm{x}_{j1})^2 + (\mathrm{x}_{i2} - \mathrm{x}_{j2})^2 + \ldots + (\mathrm{x}_{id} - \mathrm{x}_{jd})^2} \tag{2.1}$$

Let k_i to be the number of neighbors among k nearest that belong to C_i and $V^k(\mathrm{x})$ is the volume of the d-dimensional hypersphere centered at x [6]:

$$\hat{p}(\mathrm{x}|C_i) = \frac{k_i}{N_i V^k(\mathrm{x})} \tag{2.2}$$

Then we can assume that:

$$\hat{P}(C_i|\mathrm{x}) = \frac{\hat{p}(\mathrm{x}|C_i)\hat{P}(C_i)}{\hat{p}(\mathrm{x})} = \frac{k_i}{k} \tag{2.3}$$

k-NN is a typical lazy classification method that does not build a classifier until a new object needs to be classified. During classification, the distances between the new object and each object in the training set are computed. So the time complexity is $O(n)$ [7].

k is a user-defined constant and k-NN algorithm classifies an unlabeled vector by assigning the label that is the most frequent among the k training samples nearest to that point.

The best choice of k parameter depends on the data. Choosing a small k value make the k-NN more sensitive to noise and choosing a big value of k reduces the effect of noise but the boundaries between classes become less distinct. Cross-validation can be used to select a good k parameter.

ML-Lab supports resampling methods such as K-fold cross-validation and 5×2 cross-validation methods (discussed in Section 2.3). ML-Lab supports an alternative parameter type to choose the best k value, as will be shown in the experiment results chapter.

2.1.2. Naive Bayes Classification Algorithm

Naive Bayes is a simple probabilistic classifier based on Bayes' theorem, where features are assumed to be independent given the class. The assumption of independence makes it much easier to estimate these probabilities since each attribute can be treated separately. For example, an animal may be considered to be a dog if it is barking and has four legs. Even if these features depend on each other or upon the

existence of the other features, a Naive Bayes classifier considers all of these properties to independently contribute to the probability that this animal is a dog.

Naive Bayes algorithm works as follows: for each decision class it computes the conditional probability that decision class is the correct one, given an object's information vector. The algorithm assumes that the object's attributes are independent. The probabilities involved in producing the final estimate are computed as frequency counts from a "master" decision table [8].

Given the previous description of Naive Bayes, we can say that the probability of getting the string of feature values $P(X_j^1 = a_1, X_j^2 = a_2, \ldots, X_j^n = a_n|C_i)$ is just equal to the product of multiplying together all of the individual probabilities which is much easier to compute as well as reducing the curse of dimensionality: $P(X_j^1 = a_1|C_i) \times P(X_j^2 = a_2|C_i), \ldots, P(X_j^n = a_n|C_i) = \prod_k P(X_j^k = a_k|C_i)$.

Naive Bayes classifier selects the class C_i for which the following computation is the maximum [9]:

$$P(C_i|\mathrm{x}) \propto P(C_i) \prod_k P(X_j^k = a_k|C_i) \tag{2.4}$$

Despite its simplicity, Naive Bayes is successful in many applications. Its advantage is that it requires a small amount of training data to estimate the parameters necessary for classification.

2.1.3. SVM Classification Algorithm

Support Vector Machines (SVM) is a popular classification algorithm developed by Vapnik in 1995 and it is based on the Structural Risk Minimization principle [10]. The problem is to find the decision surface that can achieve maximum seperation between the two classes. Thus, SVM picks the hyperplane that the distance from the hyperplane to the nearest data point is maximized.

Figure 2.1 shows the the idea behind SVM. The dashed lines parallel to the middle line show how much we can move the decision surface without causing a misclassication of data. Margin is the distance between these parallel lines and examples closest to the decision surface are called *support vectors* that marked with circles. Hyperplane h separates the two classes with the maximum margin.

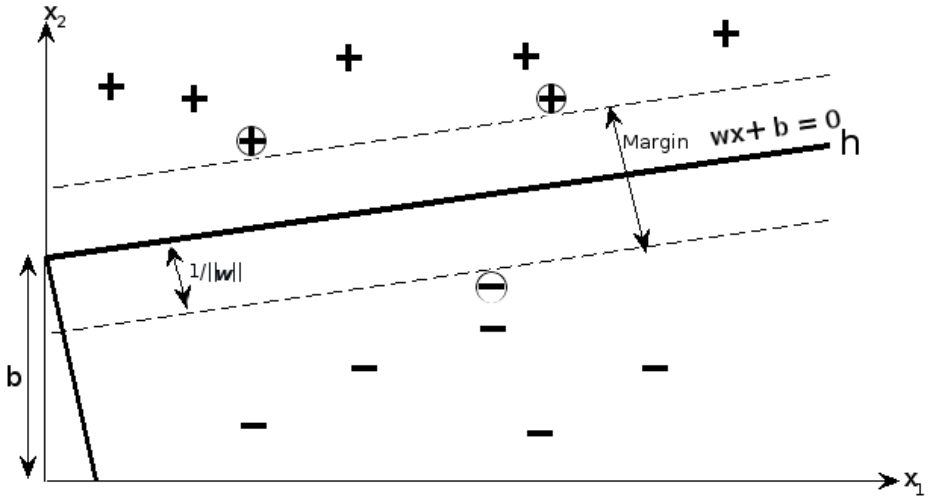

Figure 2.1. An illustration of Support Vector Machines

For the linearly separable case, we can select the two hyperplanes of the margin that there are no points between them and then try to maximize their distance. A hyperplane can be written as the set of points $\mathbf{x}$ satisfying:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.5}$$

vector $\mathbf{w}$ and constant $\mathbf{b}$ are learned from the training set.

Let $\mathrm{x}_i \in R^n$ (i = 1, 2, ..., l) and $y \in R^l$ ($y_i \in \{1, -1\}$) is our training data, The SVM problem is to find $\mathbf{w}$ and $\mathbf{b}$ that satisfy the following constraints:

$$\text{Minimize } \|\mathbf{w}\|^2 \tag{2.6}$$

$$\text{so that}, \forall i : y_i[\mathbf{w} \cdot \mathbf{x} + b] \geq 1 \tag{2.7}$$

SVM can be also used to learn non-linear decision functions. There are three different types of basis functions that are commonly used, such as polynomial, radial basis function and sigmoid. We can define these kernel function as follows [6]:

- Polynominal kernels up to some degree q in the elemens of x_k of the input vector (e.g., x_3^3 or $\mathrm{x}_1 \times \mathrm{x}_4$) with kernel:

$$K_{polynomal}(\mathbf{x}^t, \mathbf{x}) = (\mathbf{x}^T\mathbf{x}^t + 1)^q \tag{2.8}$$

  where q is selected by user.
- In this case the feature space is infinite dimensional function in the feature space Radial basis function (RBF) defines the kernel as in Parzen windows, where x^t is the center and σ defines the radius.

$$K_{rbf}(\mathbf{x}^t, \mathbf{x}) = \exp\left[-\frac{||\mathbf{x}^t - \mathbf{x}||^2}{\sigma^2}\right] \tag{2.9}$$

- Sigmoid functions:

$$K_{sigmoid}(\mathbf{x}^t, \mathbf{x}) = \tanh(2\mathbf{x}^{^T}\mathbf{x}^t + 1) \tag{2.10}$$

  where tanh(.) has the same shape with sigmoid, except that it ranges between -1 and + 1.

In our study we evaluated SVM with polynomial kernel, RBF kernel and Sigmoid kernel and we used LibSVM library implemented by [11].

Choosing which kernel to use and the parameters in these kernel is a tricky problem. The common approach is experimenting with different values and finding

one that works using validation sets and cross-validation methods that we discussed in Section 2.3.

2.1.4. C4.5 Classification Algorithm

A *decision tree* is a tree where internal nodes are tests (on input patterns) and leaf nodes are categories (of patterns). A decision tree algorithm consists of two parts: creating the tree and applying the tree to the database.

C4.5 is one of the most widely-used decision tree algorithm developed by Ross Quinlan as an extension to the earlier ID3 algorithm [12]. C4.5 has the same basic tree creation approach as ID3, but extends its capabilities to classification of continuous data by grouping together discrete values of an attribute into subsets or ranges [7]. Another advantage of C4.5 is that it can predict values for data with missing attributes based on knowledge of the relevant domains [13].

C4.5 constructs decision trees with *divide and conquer* strategy. At each node, C4.5 tries to find the locally best choice with no backtracking allowed. The pseudocode in Figure 2.2 describes the tree-construction algorithm of C4.5 [14]:

Step 1: Let T be the set of cases associated at the node. The weighted frequency, $freq(C_i, T)$ is computed of cases T whose class is C_i.

Step 2: If all cases in T belong to a same class or the number of cases in T is less than a certain value, then assign the node as a leaf and create a decision node.

Step 3: If T contains cases that belong to two or more classes, then calculate gain of each attribute.

Step 4: Select the attribute with the highest information gain.

Step 5: If the attribute is continuous, then calculate the threshold.

```
FormTree(T)
(1) ComputeClassFrequency(T);
(2) if OneClass for FewCases: return a leaf;
create a decision node N;
(3) ForEach Attribute A: ComputeGain(A);
(4) N.test = AttributeWithBestGain;
(5) if N.test is continuous: find Threshold;
(6) ForEach T' in the splitting of T:
(7) if  T' is Empty: Child of N is a leaf;
(8) else Child of N = FormTree(T');
(9) Compute Errors of N;
return N
```

Figure 2.2. Pseudocode of C4.5 Tree Construction

Step 6: A decision node has s children where $T_i, \ldots, T_s$ are the sets of splitting.

Step 7: For $i = [1, s]$, if T_i is empty, assign a leaf to the child node.

Step 8: If T_i is not empty, then apply divide and conquer, apply the same steps on the set T_i plus those cases in T with the unknown value of the selected attribute.

Step 9: Calculate classification error of the node as the sum of errors of the child nodes. If the classification error is greater than the error of classifying all cases in T, then assign the node a leaf and remove all sub-trees.

Now let us explain the *gain* and *entropy* concepts that we used in our pseudocode. *Gain* is used to rank attributes and to build decision trees where at each node we use the attribute with the greatest gain. *Entropy* is used to measure how informative a node is.

Let the training data be a set of already classified samples as $S_i = s_1, s_2, ..., s_n$ and each sample as $s_i = x_1, x_2, ..., x_n$ where $x_1, x_2, ..., x_n$ shows features of the sample.

Let S be the case set, n be the number of classes in the partition S and p_i be the proportion of S_i to S, we use the formula below to calculate the entropy:

$$Entropy(S) = \sum_{i=1}^{n} -p_i * \log_2 p_i \tag{2.11}$$

The probability p_i gives us an indication of how uncertain we are about the data and log_2 measure represents how many bits we would need to use in order to specify what the class is of a random instance. For example, the result of a fair coin toss, with the probability 0.5, can be transmitted using $-log_2(0.5) = 1$ bit, which is a zero or 1 (depending the result of the toss). Thus, we use Equation 2.11, a weighted sum of the $log_2 p_i$ for variables with several outcomes [15].

C4.5 uses the entropy as follows. Let us have a candidate split, T, which partitions S into several subsets, $\{S_1, ..., S_i, ..., S_n\}$.

$$Entropy_T(S) = \sum_{i=1}^{n} P_i Entropy_T(S_i) \tag{2.12}$$

where P_i is the proportion of records in subset i. Then, to count the gain, we use the formula below:

$$Gain(T, A) = Entropy(S) - Entropy_T(S) \tag{2.13}$$

Gain(T, A) represents the increase in information produced by partitioning the data according to the candidate split T. C4.5 algorithm chooses the optimal split to be the split that has the greatest information gain, Gain(T,A).

Let us show Figure 2.3 shows a simple decision tree T. The decision rules for tree T as follows [7]:

1. Rule 1: If (A = x_1 and B = y_1), then Classification = Class 1;
2. Rule 2: If (A = x_2 and C = z_1, then Classification = Class 2;
3. Rule 3: If (A = x_2 and C = z_2), then Classification = Class 1.
4. Rule 4: If (A = x_1 and B = y_2), then Classification = Class 2;

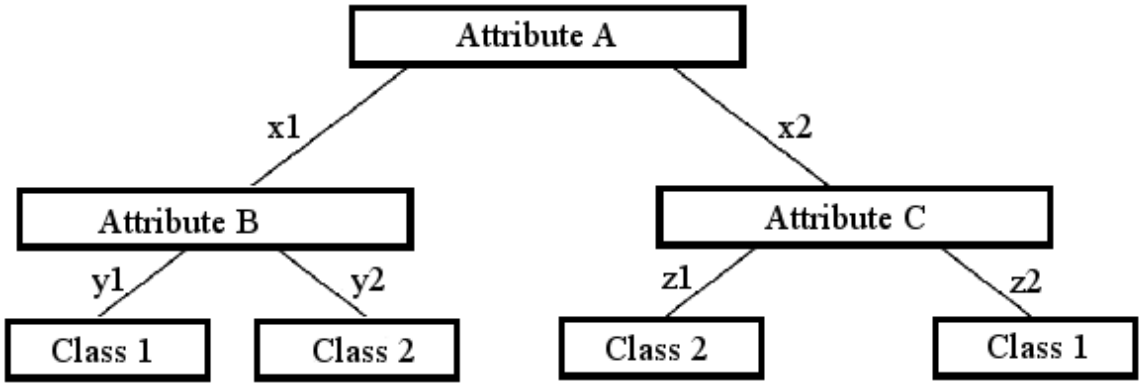

Figure 2.3. A simple decision tree T

At each iteration during the construction, C4.5 chooses the feature of the data that most efficiently divides the data into subsets that as much as possible contain instances of the same class.

2.2. Dimensionality Reduction

The *curse of dimensionality* is a common problem in machine learning. The more size and complexity of data increases, the more algorithms perform ineffectively.

Dimensionality reduction is the process of transforming the data from a high-dimensional space to a space of fewer dimensions. Preprocessing the data in order to get a smaller set of representative features not only reduces the curse of dimensionality but also the computational cost of many of the algorithms.

ML-Lab supports PCA, LDA, ISOMAP and forward feature selection algorithms. We discuss these methods briefly in the following subsections.

2.2.1. PCA

Principal Component Analysis (PCA) is a linear method for dimensionality reduction algorithm that performs a linear mapping of the data to a lower dimensional space so that the variance of the data in the low-dimensional representation is maximized.

PCA algorithm first centres the data by substracting the mean and selects the direction with the largest variation. Then it places an axis in that direction and looks at the remaining variation. Then it finds another orthogonal axis to the first and covers as much of the remaining variation as possible [9].

Let the projection of $\mathbf{x}$ on the direction of $\mathbf{w}$ be $\mathrm{z} = \mathbf{w}^T\mathbf{x}$ [6]. We need to find a $\mathbf{w}$ such that Var(z) is maximized:

$$\mathrm{Var(z)} = \mathbf{w}^T\mathbf{\Sigma}\mathbf{w} \tag{2.14}$$

where

$$\mathrm{Cov(x)} = E[(\mathrm{x} - \mu)(\mathrm{x} - \mu)^T = \Sigma \tag{2.15}$$

Maximize Var(z) subject to $||\mathbf{w}|| = 1$:

$$\max_{\mathbf{w}_1}\mathbf{w}_1^T\mathbf{\Sigma}\mathbf{w}_1 - \alpha(\mathbf{w}_1^T\mathbf{w}_1 - 1) \tag{2.16}$$

where $\mathbf{w_1}$ is the eigenvector of $\mathbf{\Sigma}$ and PCA chooses the one with the largest eigenvalue for Var(z) to be maximum [6].

2.2.2. LDA

Linear discriminant analysis (LDA) is another dimensionality reduction algorithm implemented in ML-Lab. The idea behind LDA is reducing the dimensionality while keeping as much of the class discriminatory information as possible.

As in PCA, let the projection of $\mathbf{x}$ on the direction of $\mathbf{w}$ be z = $\mathbf{w^T x}$. We need to find a $\mathbf{w}$ where z is k-dimensional and $\mathbf{w}$ is $d \times k$ [6]. The scatter matrix for C_i is:

$$\mathbf{S}_i = \sum_t r_i^t (\mathbf{x}^t - \mathbf{m}_i)(\mathbf{x}^t - \mathbf{m}_i)^T \tag{2.17}$$

where $r_i^t = 1$ if $\mathbf{x^t} \in \mathbf{C_i}$. Then we can write the total within-class scatter as:

$$S_W = \sum_{i=1}^{K} S_i \tag{2.18}$$

Let $\mathbf{m} = \frac{1}{K} \sum_{i=1}^{K} \mathbf{m_i}$ and $N_i = \sum_t r_i^t$, we can define the between-class scatter as:

$$\mathbf{S}_B = \sum_{i=1}^{K} N_i (\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T \tag{2.19}$$

Finally, we can find the matrix W that maximizes:

$$J(\mathbf{W}) = \left| \frac{\mathbf{W}^T \mathbf{S}_B \mathbf{W}}{\mathbf{W}^T \mathbf{S}_W \mathbf{W}} \right| \tag{2.20}$$

2.2.3. ISOMAP

ISOMAP is one of the popular low-dimensional embedding methods. It is developed by Tenenbaum et al [16] in 2000. ISOMAP uses geodesic distances on a weighted graph together with metric multidimensional scaling.

Isomap defines the geodesic distance to be the sum of edge weights along the shortest path between two nodes (computed using Dijkstra's algorithm [17]). It then uses multidimensional scaling to map the true geodesic distances to coordinates in a low-dimensional space.

2.2.4. Forward Feature Selection

Forward feature selection (FFS) is a subset selection method that starts with a null feature-subset and each step, it adds one feature that decreases the error most. It continues until any further addition does not decrease the error. Pseudo algorithm of forward selection is as follows [18]:

$S^t \leftarrow \emptyset$

repeat

$j \leftarrow \arg\, max_j\ q(S^t \cup \{j\})$

$S^t \leftarrow S^t \cup j$

$S^t \leftarrow S \setminus j$

until $S = \emptyset$

Figure 2.4. Pseudocode of Forward Feature Selection

2.3. Resampling

We cannot use the same data both for training and testing and with small data sets, we need resampling methods to be able to get sufficiently large training and validation sets for a full data set.

ML-Lab supports Training/Validation, K-fold Cross Validation and 5×2 Cross Validation Methods. To have more reliable results, ML-Lab supports using a separate set for training the classifier and using a separate set for calculating the error rates. The training dataset is used to train or build a model. Once a model is built on training data, we need to find out the accuracy of the model on unseen data. For this purpose, the model should be used on a dataset that was not used in the training process, a dataset where we know the actual value of the target variable.

2.3.1. K-fold Cross Validation

K-fold cross validation is a resampling method. While Training/Validation method causes more accurate results, it reduces the amount of data that is available for testing. K-fold cross validation method divides a data set into k mutually exclusive partitions of equal size. The classifier trains k times by using one of the sets as the validation set and the rest of sets as the training set in each time.

The average error of k partitions is called the cross-validated error rate.

2.3.2. 5×2 Cross Validation

5×2 Cross Validation is very similar to K-fold Cross Validation, but it performs five replications of 2-fold cross validation by dividing the dataset into two equal-sized sets in each replication.

2.4. Evaluation

We discussed different machine learning methods; classification algorithms and dimensionality reduction algorithms. In this section, we will discuss how to report and analyze the results. The error and performance of a method can be evaluated by comparing the prediction with the actual values.

ML-Lab supports calculating the Confusion Matrix, ROC Curve and PR Curve.

2.4.1. Confusion Matrix

A confusion matrix contains information about the actual and predicted classifications done by the classification algorithm. Each row of the confusion matrix represents the predictions while each column represents the instances in the actual class (see Table 2.1).

Table 2.1. Confusion Matrix

True Class	Positive	Negative	Total
Positive	TP	FN	P
Negative	FP	TN	N

Accuracy (AC) is the proportion of the total number of predictions that are correct. It is determined using the equation:

$$accuracy = \frac{TP + TN}{P + N} \tag{2.21}$$

True positive (TP): Number of instances for which both the class label and the predicted class are positive.

False negative (FN): Number of instances for which the class label is positive and the predicted class is negative.

False positive (FP): Number of instances for which the class label is negative and the predicted class is positive.

True negative (TN): Number of instances for which both the class label and the predicted class are negative.

Figure 2.5 shows the confusion matrix on the Mfeat dataset using 5-NN algorithm.

2.4.2. ROC Curve

Receiver Operator Characteristic (ROC) curves are commonly used to evaluate the performance in binary classification problems. ROC curves represent the percentage of **true positives** on the y axis and **false positives** on the x axis.

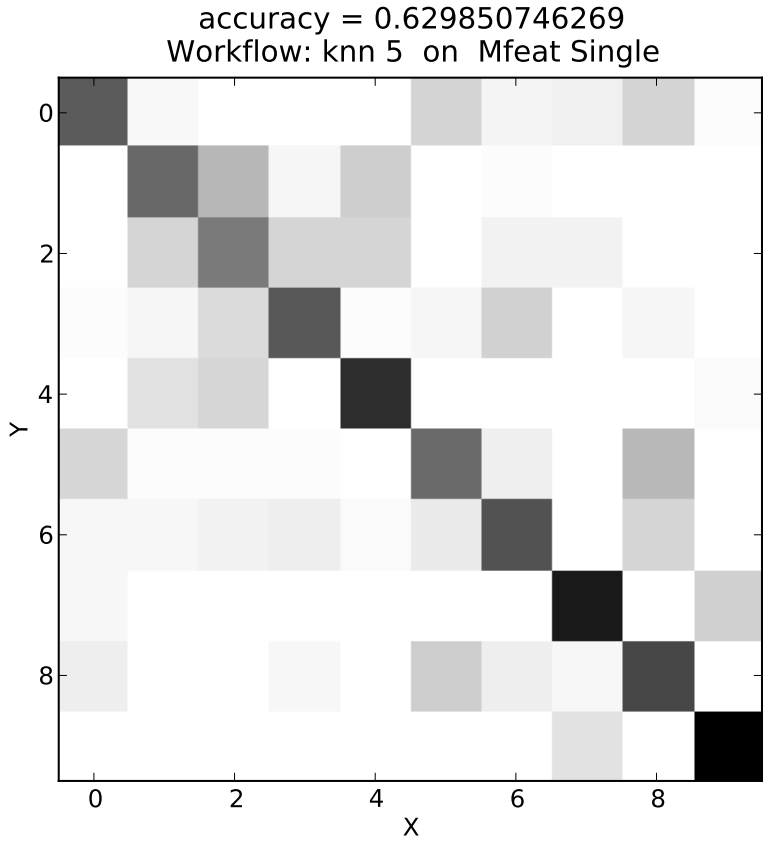

Figure 2.5. A Sample Confusion Matrix for Mfeat Dataset

True positive rate is generally called **specificity** while false negative rate is called **sensitivity**.

$$Specificity = TN/N = TN/(FP + TN) \tag{2.22}$$

$$Sensitivity = TP/P = TP/(TP + FN) \tag{2.23}$$

The closer the curve follows the left-hand border and the top border of the ROC space, the more accurate is the classifier. ROC curves are commonly used with cross validation method rather than a point. ML-Lab supports K-fold cross validation and 5×2 fold cross-validation methods for ROC curves. For example using a 10-fold cross-validation, there will be 10 different ROC curves that correspond to 10 different test sets.

The Area Under the Curve (AUC) is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one [19]. An example is given in Figure 2.6.

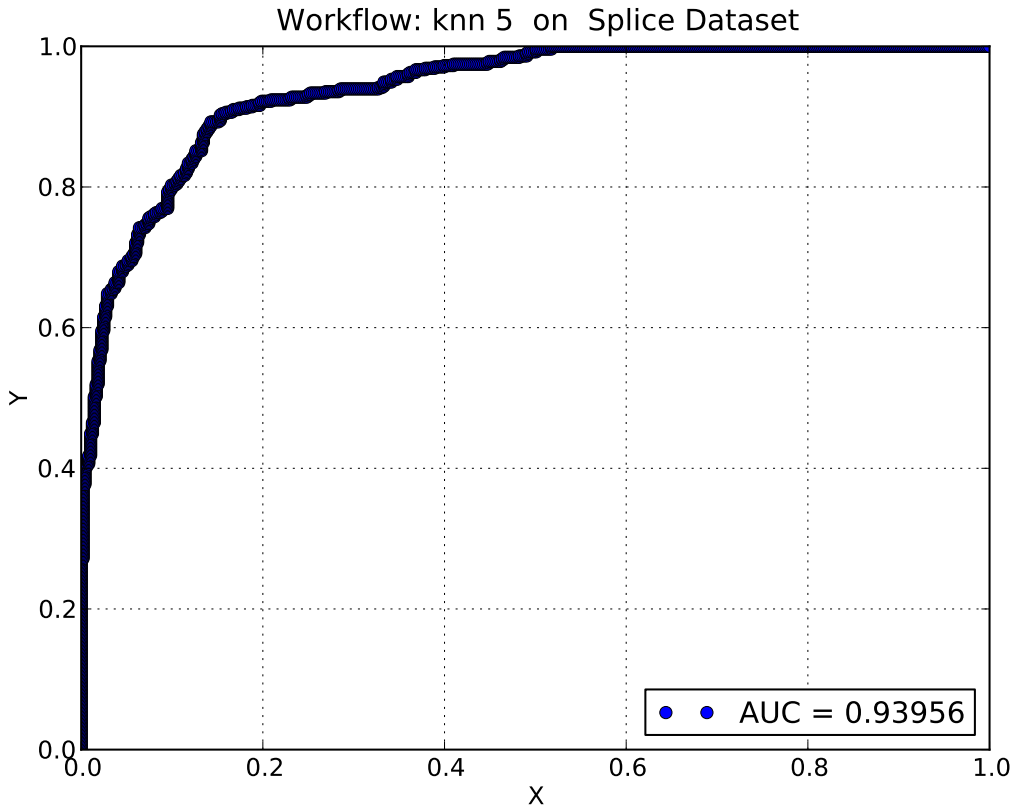

Figure 2.6. A Sample ROC Curve for Splice Dataset

2.4.3. PR Curve

Precision-Recall (PR) curves is a plot of the true positive rate (so called, recall) on the x-axis, and the precision on the y-axis.

The closer curve follows the upper-right portion of the graph, the more accurate is the test. When the curve is in the lower-left portion of the graph, the classifier's performance is low.

$$Precision = \frac{TP}{TP + FP} \tag{2.24}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.25}$$

An example is given in Figure 2.7.

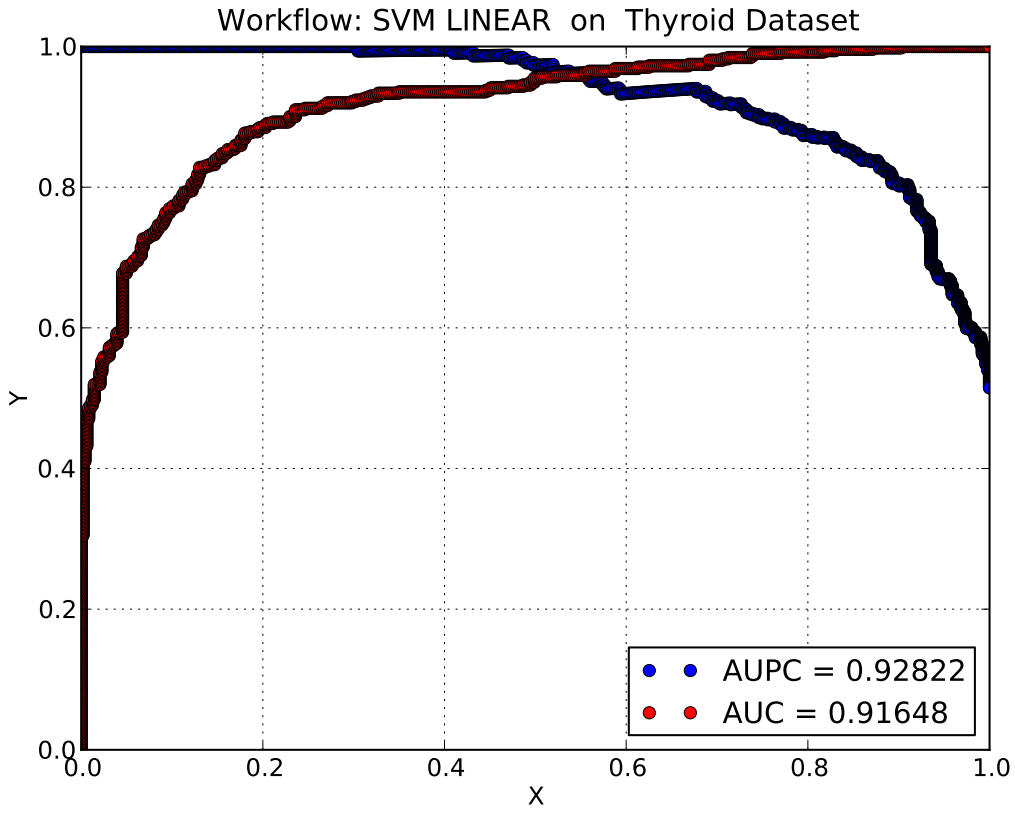

Figure 2.7. A Sample PR/ROC Curve for Thyroid Dataset

2.4.4. Hypothesis Testing

Hypothesis testing is used to test a particular hypothesis concerning the parameters. A result can be assumed to be significant if it is unlikely to have occurred by chance. ML-Lab supports Paired t Test, 5×2 Paired t Test and 5×2 Paired F Test methods.

In Paired t test, we the algorithm K times, on K training/validation set pairs, we get K error percentages. Let $x_i^t = 1$ if the classifier makes a misclassification error on

instance t of V_i, $\mathrm{x}_i^t = 0$ otherwise [6]. Assume that p_i is the error percentage:

$$p_i = \frac{\sum_{t=1}^{N} x_i^t}{N} \tag{2.26}$$

Then,

$$m = \frac{\sum_{i=1}^{K} p_i}{K}, S^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K - 1} \tag{2.27}$$

Finally we get the following formula for Paired t-test:

$$\frac{\sqrt{K}(m - p_0)}{S} \sim t_{K-1} \tag{2.28}$$

We reject the null hypothesis that the classification algorithm has higher error percentage than p_0 at significance level α if this value is higher than $t_{\alpha,K-1}$.

2.5. INTEGRATION METHODS

We covered general classification approaches in Section 2.1. In all these approaches, we used a single set of data for training and a single classification method to produce a classifier. Although there is no single *best* method for all classification problems, we can always find the *best* classification method for a given data set. Integration methods aims to reach an overall better performance than could be achieved by using each of them separately.

ML-Lab supports Early Integration method and Late Integration Method.

In *Early integration*, two or more types of data (e.g. different features of a dataset) are concatenated to form a single set of input vectors and then we give it this large input vector to a single classification algorithm. Figure 2.8 shows an illustration of Early integration [20].

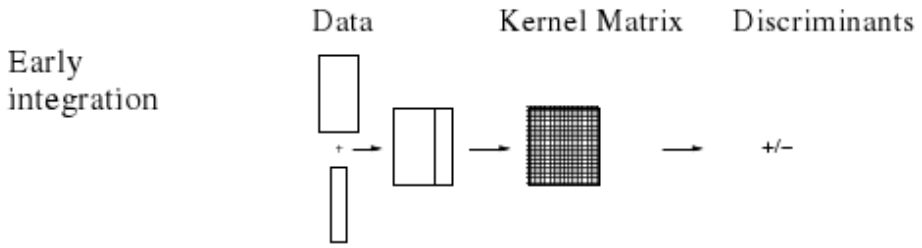

Figure 2.8. An illustration of Early Integration with Support Vector Machines

Late Integration uses a single dataset and trains multiple classifiers on it and then combines their decisions by a single classification algorithm. Figure 2.9 shows an illustration of Late integration [20]. One SVM is trained on each data type, and the resulting discriminant values are summed.

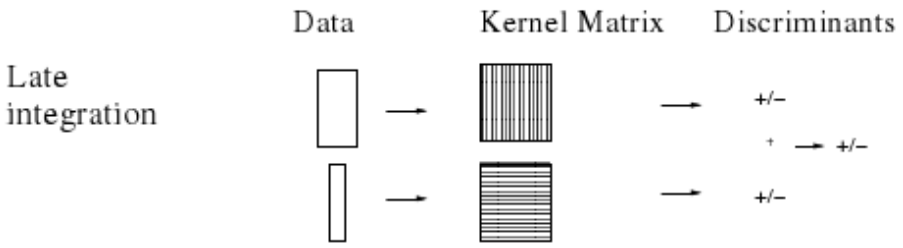

Figure 2.9. An illustration of Late Integration with Support Vector Machines

3. ARCHITECTURE

In this chapter, we discuss ML-Lab's architecture. ML-Lab consists of two main parts: the graphical user interface and the backend. The graphical user interface part is responsible for interacting with the users and providing the connection between the users and the backend. The user interface of ML-Lab consists of four different components; Dataset, Classification, Dimensionality Reduction and Visualization. The backend of ML-Lab consists of two different parts; libraries and engines. Engines are responsible for processing the workflows by using the libraries and returning the results to the graphical interface. A screenshot of ML-Lab is given in Figure 3.1.

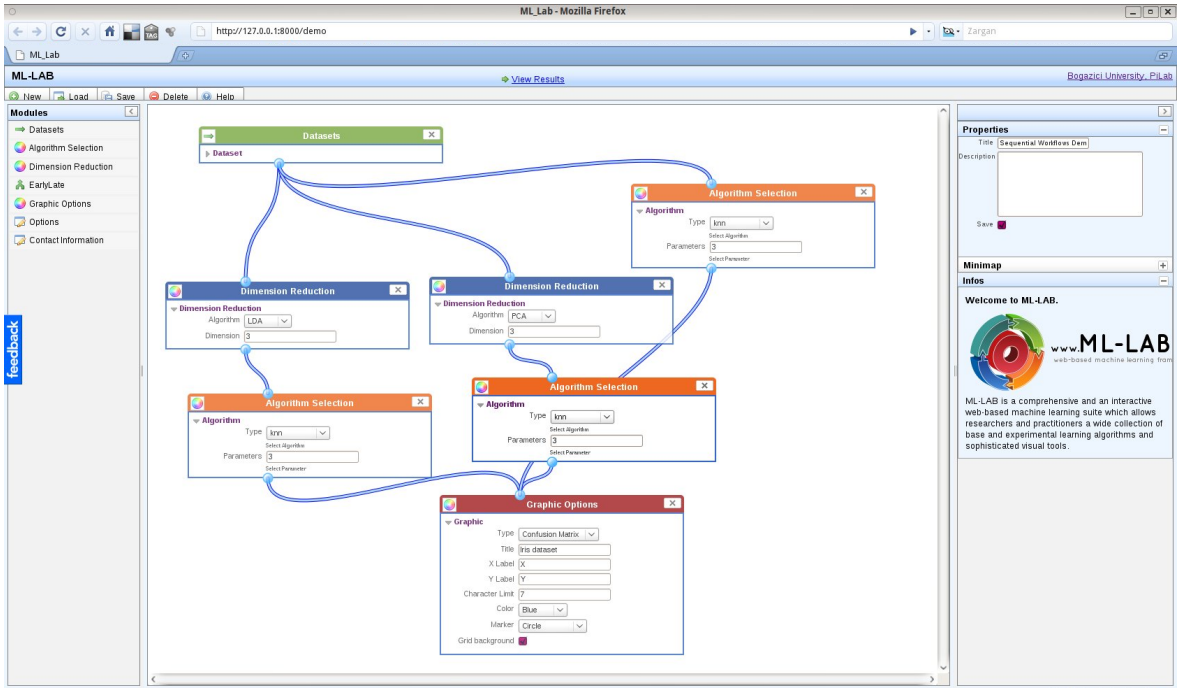

Figure 3.1. Main screen of ML-Lab with a sample workflow

3.1. Web Interface

ML-Lab's web interface is responsible for the connection between the user and the backend. ML-Lab's interface is designed as a user-friendly interface and consists of connectable boxes. Each box can be connected together (with the exception of Dataset

component) with the help of wires and connected components become workflows. When a user decides to run the workflow, the graphical user interface component delivers the workflow with the parameters and the values and sends them to the backend.

ML-Lab's web interface is written in Django Framework [21] and WireIt library [22].

3.1.1. Dataset Component

Dataset component is the only component that is not connectable to other components. It is responsible for all of the resampling, converting and processing issues. The Dataset component can be used in three different ways and can take two types of input. There are three ways to give an input to ML-Lab:

1. Dataset component can take a single set as an input. In this case, ML-Lab will train and test on the same examples.
2. Dataset component can take a training set and a validation set. In this case, ML-Lab will train the classifiers from the training set, and will predict the results from the validation set.
3. K-fold Cross Validation: ML-Lab can take a K parameter and run K-fold cross validation algorithm.

There are three different data types available in ML-Lab:

1. Numeric data: The default format of ML-Lab.
2. Discrete data: If the given data is discrete, ML-Lab converts it into a numeric dataset.
3. Kernel Matrix: ML-Lab allows to use kernel matrices for the SVM classifier.

3.1.2. Classification Component

The Classification component takes the preprocessed data from the Dataset component and runs the selected algorithm with a given set of parameters. This component interacts with any type of components (e.g. taking a component as an input for itself or giving an output to be another component's input).

Classification component supports k-NN, Naive Bayes, SVM and C4.5 classification algorithms, as we have discussed earlier in Section 2.1.

3.1.3. Dimensionality Reduction Component

Dimensionality Reduction Component takes the preprocessed data from the Dataset component and runs the selected algorithm with the given parameters. ML-Lab supports LDA, PCA, Isomap and Forward Feature Selection methods which we discussed earlier in Section 2.2.

3.1.4. Visualization Component

This component takes graphic type, color, marker type and various options (grid background, transparent background or landscape orientation), plots the dataset and return the graphic as in various formats: EPS, PS, PDF and PNG.

The color, marker and other available options are given in Appendix A.

ML-Lab currently supports Confusion Matrix, ROC Curve, PR Curve and 2D/3D plotting and Comparison Table in Latex format. Confusion Matrix, ROC Curve and PR Curve have been discussed in Section 2.4. Now we will introduce the 2D/3D plotting and Comparison Table visualization options.

3.1.4.1. 2D Plotting. 2D Plotting option is used after reducing the dimensionality of a data to two. Figure 3.2 shows an example of two-dimensional representation of Mfeat Dataset.

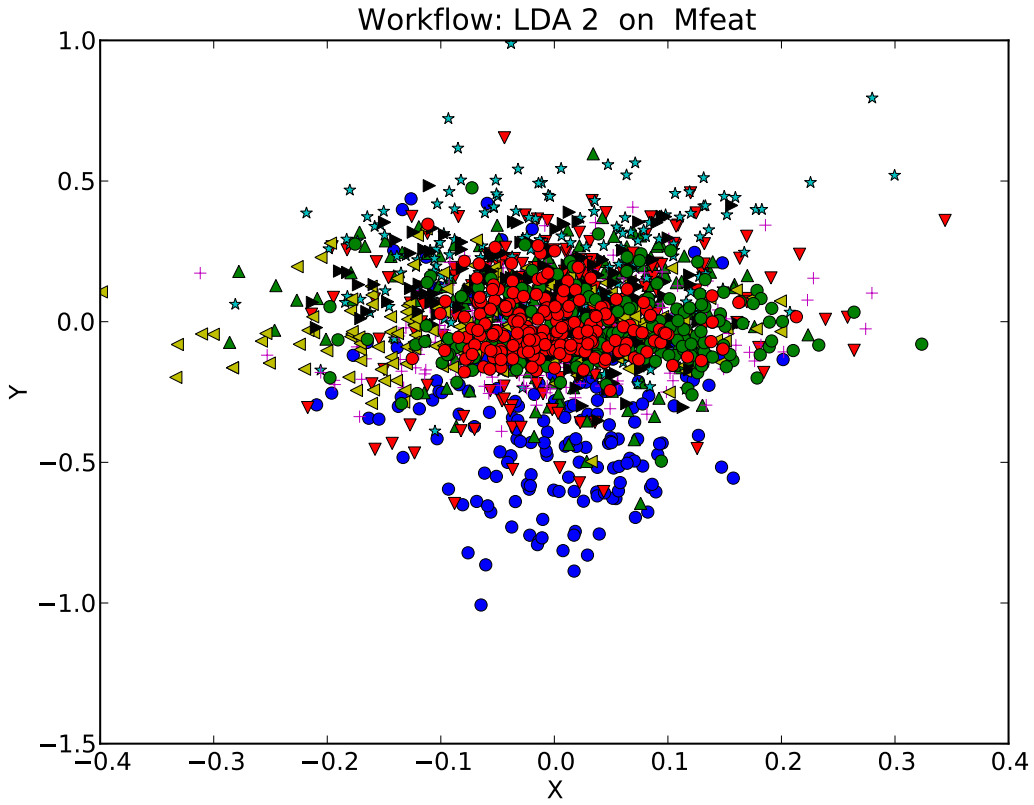

Figure 3.2. A Sample 2D Plotting for Mfeat Dataset

3.1.4.2. 3D Plotting. 3D Plotting option is used after reducing the dimensionality of a data to three. Figure 3.3 shows an example of three-dimensional representation of the Mfeat Dataset.

3.1.4.3. Comparison Tables with Latex. ML-Lab can also generate comparison tables in .tex format to allow users to embed them in their own documents (see Table 3.1).

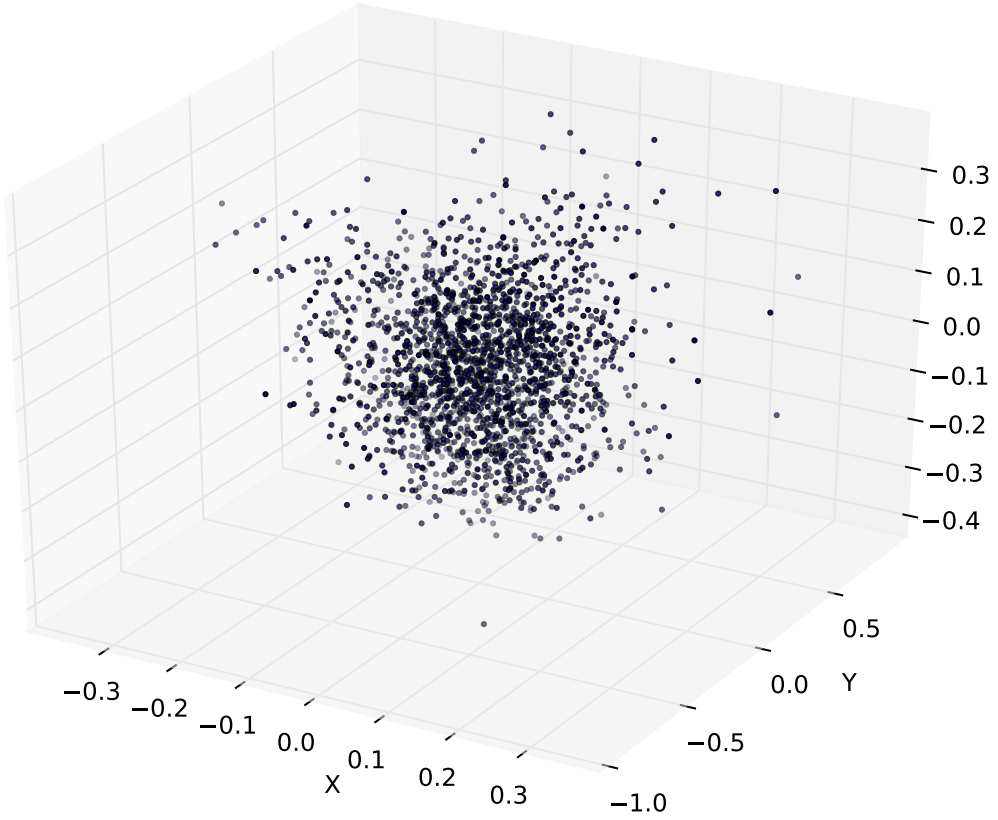

Figure 3.3. A Sample 3D Plotting for Mfeat Dataset

Table 3.1. Comparison Table for Splice Dataset

Results	5-NN	Naive Bayes	SVM
Accuracy	0.79798	0.91658	0.84422
AUC	0.93956	0.97321	0.91648
AUPC	0.94160	0.97310	0.92822

3.2. Backend

ML-Lab's backend is designed as a library and a set of engines for complicated tasks. In this section, we will discuss the different libraries and engines of ML-Lab.

3.2.1. Machine Learning Library

ML-Lab's machine learning library consists of all of the classification and dimensionality reduction algorithms that we discussed in earlier chapters. This library is designed separately from ML-Lab, thus anyone can download this package and use it for their own projects (Appendix B).

ML-Lab's machine learning library is written in Python language with scientific packages; Numpy and Scipy [23].

3.2.2. Graphical Library

ML-Lab's graphical library consists of procedures for generating a confusion matrix, ROC/PR Curves, 2D/3D plotting methods and can be used seperately from ML-Lab. In Appendix 3, we explain how to use this graphical library in Python.

ML-Lab's graphical library is written in Python language with Matplotlib, a 2D graphics package used for Python for application development and interactive scripting [24].

3.2.3. Workflow Generator Engine

Workflow generator solves the dependency issues between the connected components and constructs a tree structure with ancestors and nodes that shows which component to be processed after which. After generating the workflow, it sends a request to the Workflow Processor Engine to process the data.

3.2.4. Workflow Processor Engine

Workflow processor takes a workflow as an input and processes tasks one after the other. Each task output is an input for the next task. There is no limit for the number

of tasks, thus it is possible to create dozens of sequential workflows for experimental uses.

3.2.5. Save/Load Workflow Engine

ML-Lab allows users to save and load workflows. Save/Load Workflow engine allows users to save the current workflow from the user interface and can load it at a later time.

Workflows can also be downloaded to a local disk and can be uploaded to the server at any time. Thus, users can continue to work on the same workflow.

Each workflow in ML-Lab can be reached using its id, e.g.:
http://www.ml-lab.com/graphic/533

3.2.6. Document Creator Engine

Document Creator Engine converts the graphical results into various document formats: PS, EPS, PDF and SVG.

3.2.7. Partitioner Engine

This module creates cross fold partition sets from an input sequence while taking the stratification issue into account. Thus, Partitioner Engine makes sure that the classes are represented in the right proportions.

3.2.8. Mailing Engine

Users can optionally enter their e-mail address to the Contact Information component. In this case, ML-Lab's e-mail server sends the graphics (in EPS or other desired document formats) and the url of the workflow to the entered e-mail address.

4. EXPERIMENTS AND RESULTS

In this chapter we first describe the datasets we used in our experiments and next, we present and evaluate the experimental results for the ML-Lab's algorithms.

4.1. Datasets

In this thesis, demonstrations are done with a multivariate dataset; Multiple Features and two bioinformatics datasets; Thyroid and Splice.

4.1.1. Multiple Features Dataset

Multiple Features Dataset (mfeat) consists of features of handwritten numerals ('0'–'9') extracted from a collection of Dutch utility maps. 200 patterns per class (for a total of 2,000 patterns) have been digitized in binary images. These digits are represented in terms of the following six feature sets:

- mfeat-fou: 76 Fourier coefficients of the character shapes;
- mfeat-fac: 216 profile correlations;
- mfeat-kar: 64 Karhunen-Love coefficients;
- mfeat-pix: 240 pixel averages in 2 x 3 windows;
- mfeat-zer: 47 Zernike moments;
- mfeat-mor: 6 morphological features.

4.1.2. Splice Dataset

Splice is a dataset containing primate splice-junction gene sequences (DNA) with associated imperfect domain theory. Splice junctions are points on a DNA sequence at which 'superfluous' DNA is removed during the process of protein creation in higher organisms.

The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out). This problem consists of two subtasks: recognizing exon/intron boundaries (referred to as EI sites), and recognizing intron/exon boundaries (IE sites).

The Splice dataset consists of 512 NB and 483 BD samples.

4.1.3. Thyroid Dataset

The Thyroid–problem task is to decide whether a patient has thyroid over–, normal– or underfunction. There are 21 feature, 15 of them are binary and 6 of them are continuous ones. The class probability for the normalfunction is rather high 92.6% and there are only few samples for disfunctions of the thyroid.

The Thyroid dataset consists of 62 Hyperthroid and 2723 Negative samples.

4.2. Comparison of Different Classification Algorithms

ML-Lab support k-NN, Naive Bayes, SVM and C4.5 classification algorithms. Figure 4.1 shows the workflow of how to compare different classification algorithms in one comparison table (see Table 4.1).

Table 4.1. Comparison Table for Mfeat Training/Validating Dataset

Results	7-NN	Naive Bayes	C4.5	SVM
Accuracy	0.9164	0.8955	0.9044	0.9641

AUC and AUPC values cannot be shown since it is a multiclass dataset.

Table 4.2 shows the comparison table for Splice dataset and Table 4.3 shows the comparison table for Thyroid dataset.

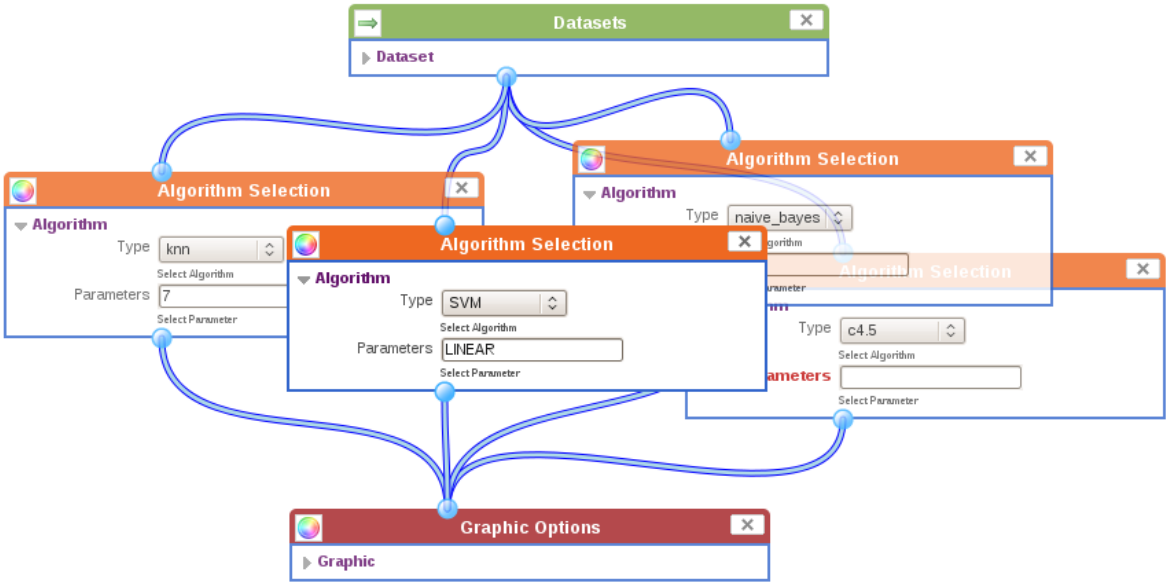

Figure 4.1. Comparison of Different Classification Algorithms

Table 4.2. Comparison Table for Splice Dataset

Results	5-NN	Naive Bayes	SVM
Accuracy	0.7979	0.9165	0.8442
AUC	0.9395	0.9732	0.9164
AUPC	0.9416	0.9731	0.9282

Table 4.3. Comparison Table for Thyroid Dataset

Results	7-NN	Naive Bayes	SVM
Accuracy	0.9856	0.9992	0.9906
AUC	0.9944	0.9999	0.9963
AUPC	0.7488	0.9801	0.8124

4.3. Comparison of Different Dimensionality Reduction Algorithms

ML-Lab supports LDA, PCA and Isomap dimensionality reduction algorithms. Figure 4.2 shows how to compare different dimensionality reduction algoritms. In

this comparison, LDA and PCA algorithms are applied to Splice dataset to reduce the dimensionality to two and plot the results. Figure 4.3 and Figure 4.4 shows the graphical results of LDA and PCA algorithms.

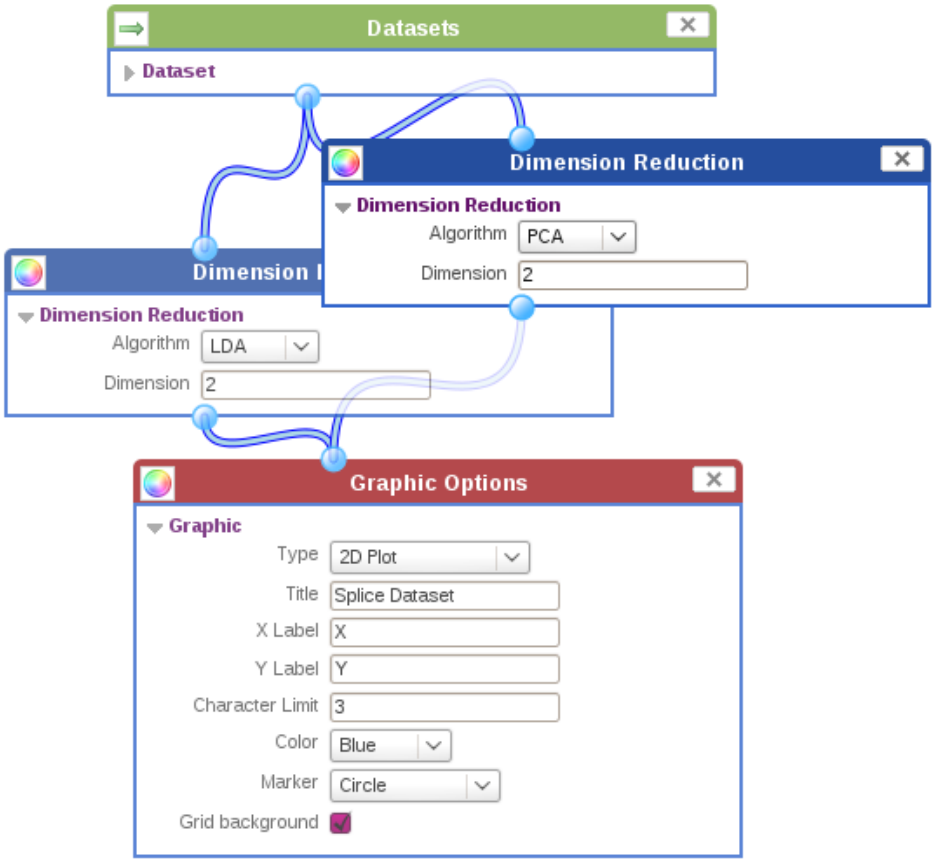

Figure 4.2. Comparison of Different Dimensionality Reduction Algorithms

After changing the parameter of Dimensionality Reduction box of LDA and PCA to three and graphic type to 3D Plotting, Figure 4.5 and Figure 4.6 shows the 3D plots of LDA and PCA.

4.4. Comparison of Sequential Workflows

ML-Lab allows users to create sequential workows, that is, to apply classification after reducing the dimensionality, or to apply classification algorithms one after one. Figure 4.7 shows the workflow of comparing the results of applying classification after LDA, applying classification after PCA, and directly applying classification.

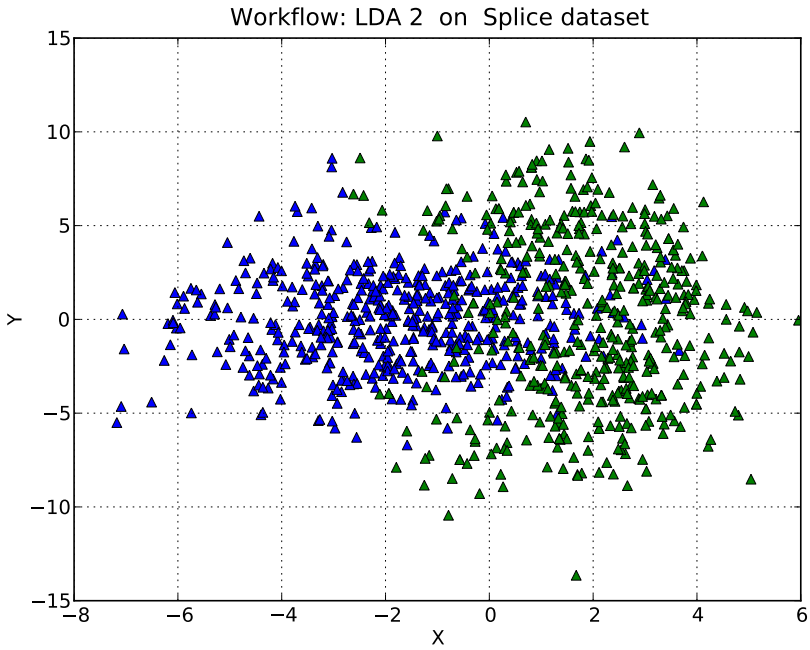

Figure 4.3. LDA with 2 Dimension on Splice Dataset

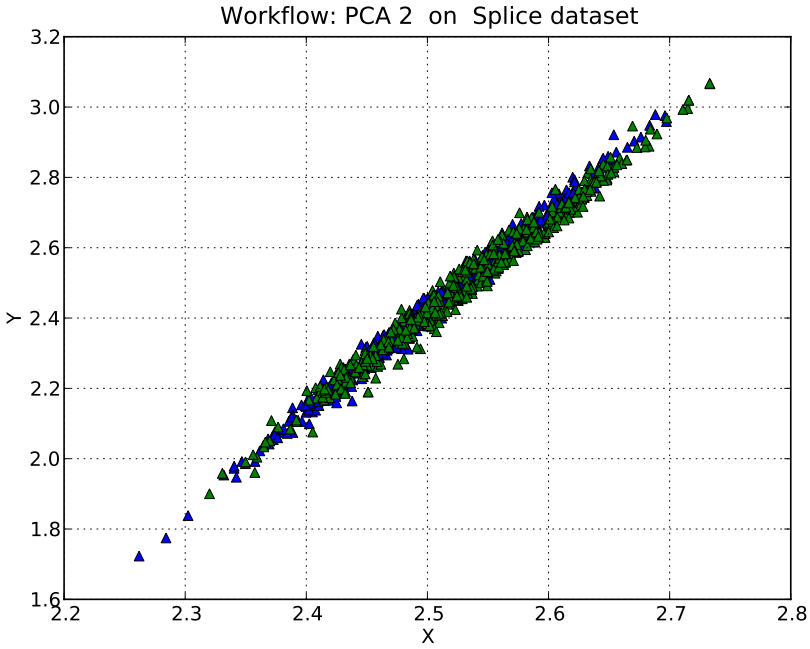

Figure 4.4. PCA with 2 Dimension on Splice Dataset

Table 4.4 shows the comparison table of sequential workflows for Mfeat dataset, Table 4.5 shows the comparison table for Splice dataset and Table 4.6 shows the comparison table for Thyroid dataset.

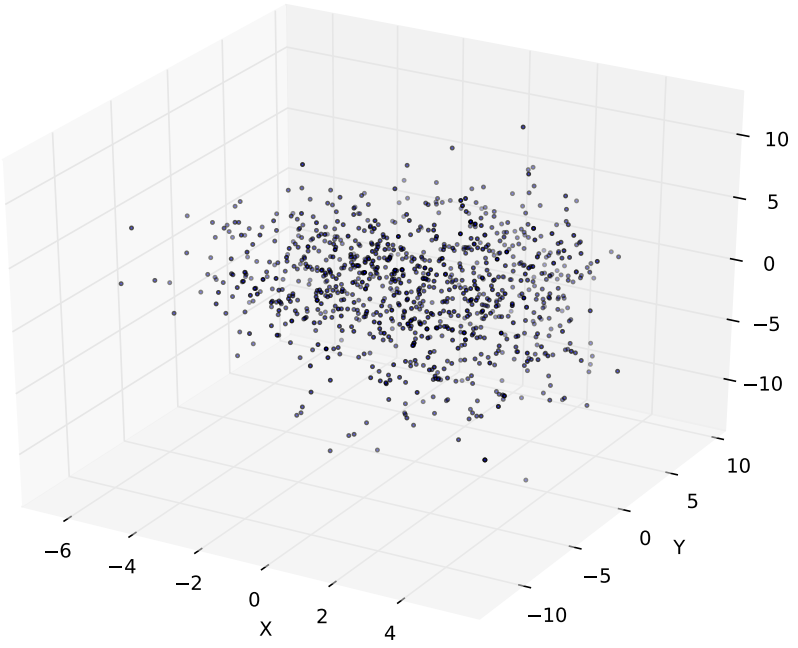

Figure 4.5. LDA with 3 Dimension on Splice Dataset

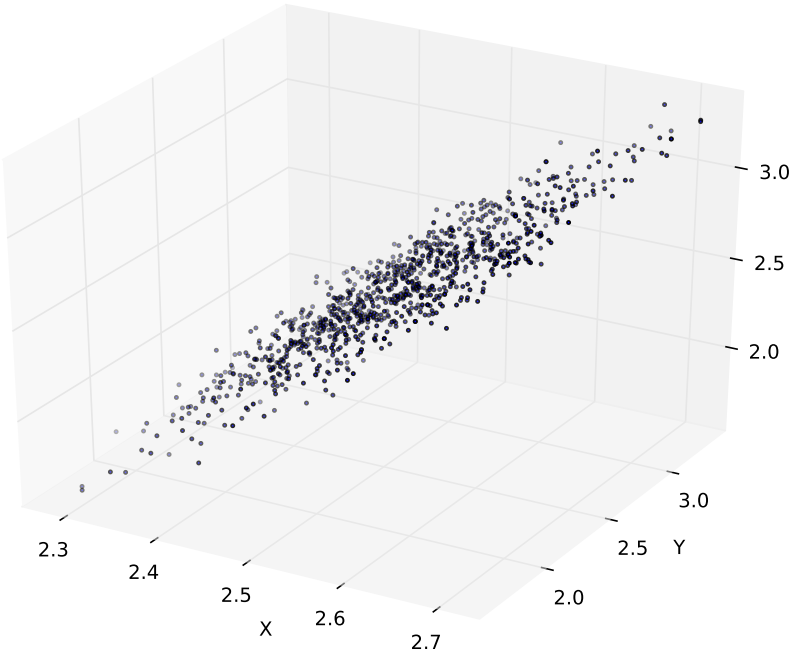

Figure 4.6. PCA with 3 Dimension on Splice Dataset

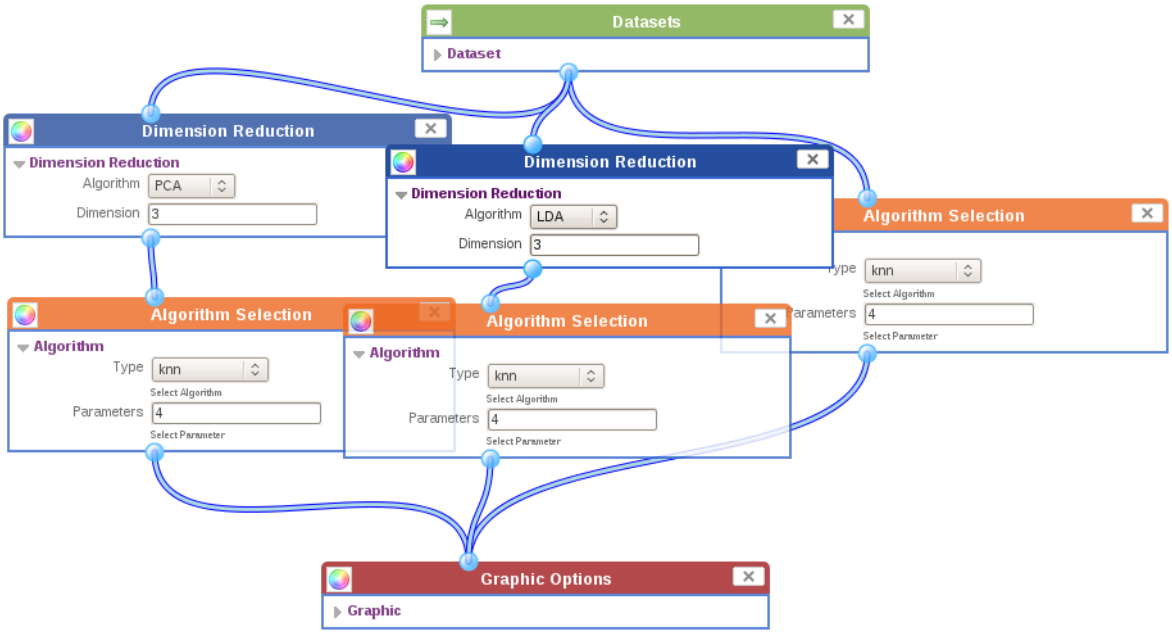

Figure 4.7. Comparison of Sequential Workflows

Table 4.4. Comparison Table of Sequential Workflows for Mfeat Dataset

Results	4-NN	LDA 3 and 4-NN	PCA 3 and 4-NN
Accuracy	0.9454	0.8469	0.7984

Table 4.5. Comparison Table for Splice

Results	3-NN	LDA 3 and 3-NN	PCA 3 and 3-NN
Accuracy	0.8442	0.9246	0.8281
AUC	0.9541	0.9838	0.9142
AUPC	0.9522	0.9836	0.9234

4.5. Comparison of Different k-NN Algorithms

ML-Lab supports a special parameter format for k-NN algorithm. In Figure 4.12, we will use **first-last, count** format as 2-6,1 which will help us to compare 2-NN, 3-NN, 4-NN, 5-NN and 6-KNN algorithms respectively.

Table 4.6. Comparison Table for Thyroid

Results	3-NN	LDA 3 and 3-NN	PCA 3 and 3-NN
Accuracy	0.9899	0.9805	0.9892
AUC	0.9969	0.9812	0.9958
AUPC	0.8610	0.4955	0.8190

Table 4.7 shows the comparison table for Splice dataset, Table 4.8 shows the comparison table for Thyroid dataset for different k-NN algorithms. Figure 4.8 shows the ROC Curves and Figure 4.9 shows the PR Curves with different k-NN algorithms on Splice dataset, Figure 4.10 shows the ROC Curves and Figure 4.11 shows the PR Curves with different k-NN algorithms on Thyroid dataset. Thyroid dataset contains 62 Hyperthroid and 2723 Negative samples where Splice dataset has 512 NB and 483 BD samples. The imbalance between the class samples of Thyroid dataset causes AUPC values to decrease faster than AUC values. This is because of specificity and precision values. As discussed in Section 2.4, specificity value is calculated with true negative (TN) and false positive (FP) values where precision is calculated with true positive (TP) and false positive (FP) values. Because of the imbalance of the Thyroid dataset, the change in TN value is much more smaller than TP value. For example, the change of TN value between 4-NN and 5-NN algorithm is 0.2% where the change of TP value is 15%. This change causes precision to decrease faster.

Table 4.7. Comparison Table for Splice

Results	KNN#2	KNN#3	KNN#4	KNN#5	KNN#6
Accuracy	0.9366	0.8442	0.8703	0.7979	0.8371
AUC	0.9714	0.9541	0.9467	0.9395	0.9342
AUPC	0.9684	0.9522	0.9451	0.9416	0.9370

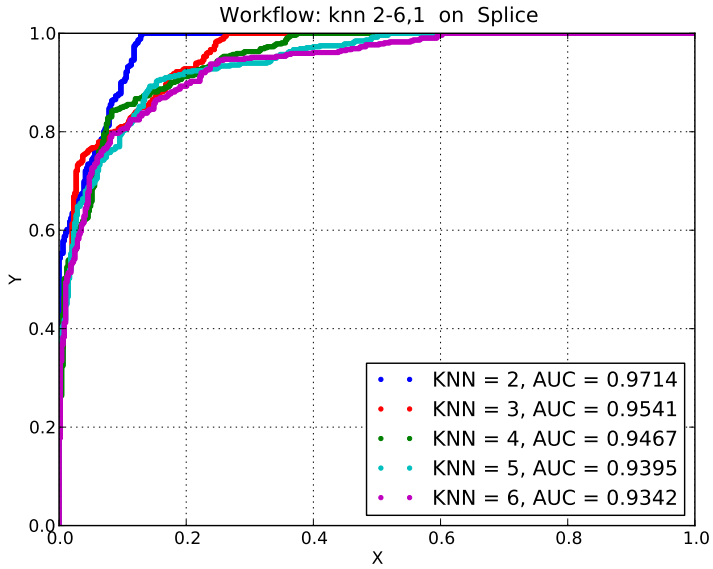

Figure 4.8. Comparison of Different k-NN Algorithms and ROC Curves on Splice Dataset

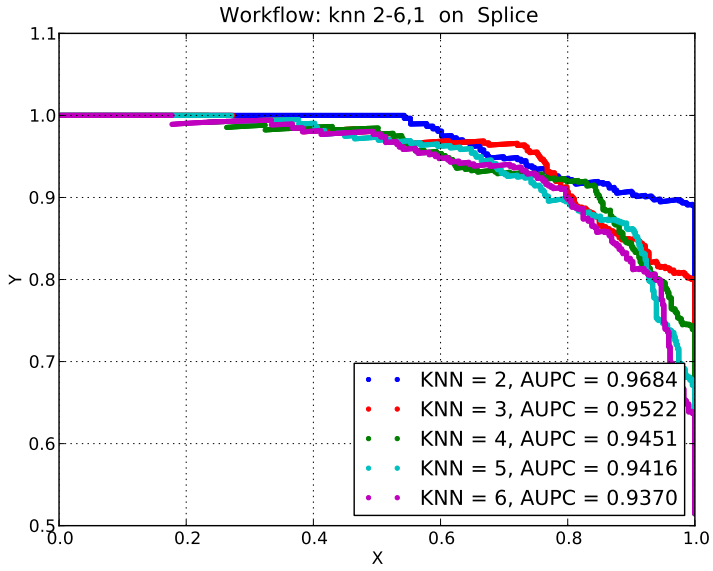

Figure 4.9. Comparison of Different k-NN Algorithms and PR Curves on Splice Dataset

4.6. K-fold Cross Validation

ML-Lab supports K-Fold Cross Validation technique for estimating the performance and we will compare results for 5-Fold Cross Validation. Table 4.9 shows the

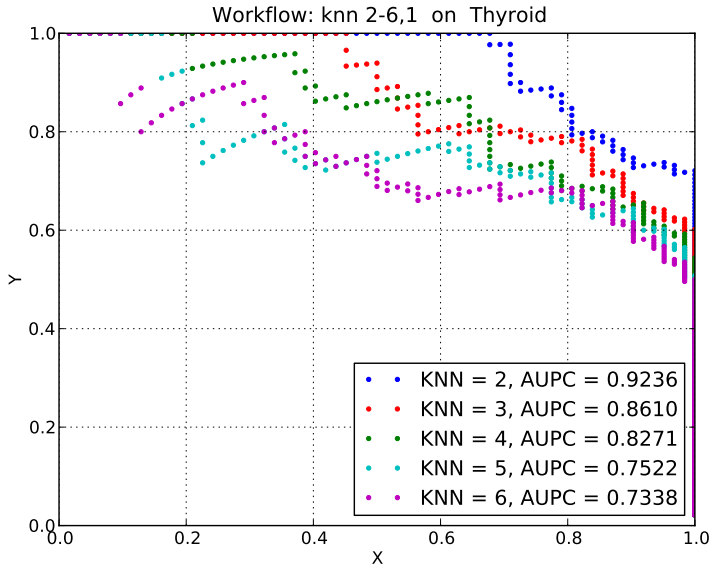

Figure 4.10. Comparison of Different k-NN Algorithms and ROC Curves on Splice Dataset

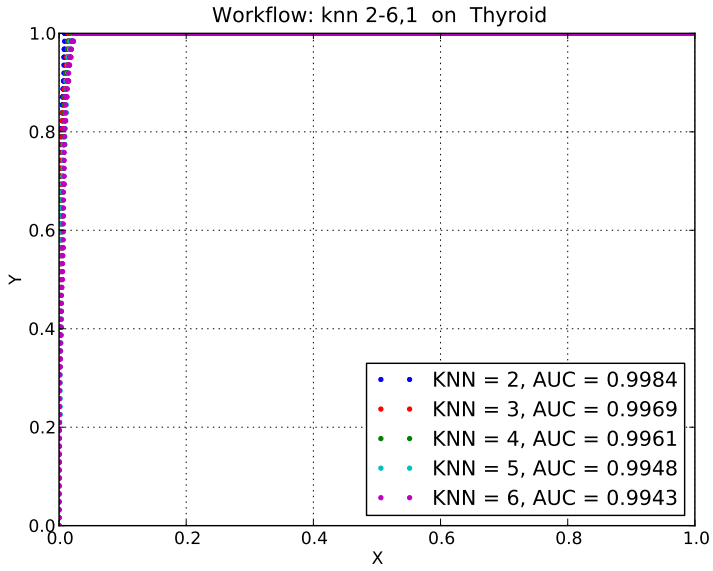

Figure 4.11. Comparison of Different k-NN Algorithms and PR Curves on Thyroid Dataset

comparison table for Splice dataset and Table 4.10 shows the comparison table for Thyroid dataset.

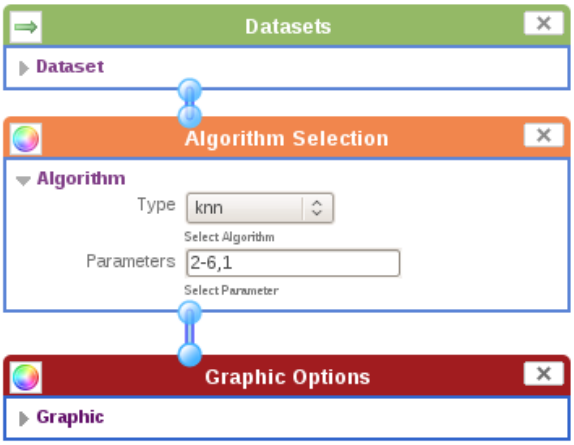

Figure 4.12. Different k-NN Algorithms

Table 4.8. Comparison Table for Thyroid

Results	KNN#2	KNN#3	KNN#4	KNN#5	KNN#6
Accuracy	0.9895	0.9899	0.9877	0.9874	0.9870
AUC	0.9984	0.9969	0.9961	0.9948	0.9943
AUPC	0.9236	0.8610	0.8271	0.7522	0.7338

K-fold Cross Validation with Thyroid dataset also suffers from the imbalance of samples as we discussed in Section 4.6. The AUPC value is much more smaller with K-fold Cross Validation, because folds contain a very few amount of hyperthyroid samples to learn.

Table 4.9. Comparison Table for Splice

Results	K-Fold#1	K-Fold#2	K-Fold#3	K-Fold#4	K-Fold#5
Accuracy	0.63	0.7	0.76	0.67	0.7
AUC	0.8375	0.7655	0.8335	0.6906	0.8345
AUPC	0.8742	0.8031	0.7479	0.5714	0.8487

Table 4.10. Comparison Table for Thyroid with 3-NN algorithm

Results	K-Fold#1	K-Fold#2	K-Fold#3	K-Fold#4	K-Fold#5
Accuracy	0.983	0.985	0.983	0.973	0.982
AUC	0.855	0.961	0.987	0.926	0.870
AUPC	0.477	0.334	0.519	0.441	0.401

4.7. Comparison of Two Forward Feature Selection

Figure 4.13 shows the workflow of how to compare two Forward Selection methods on the Iris Dataset. The first Feature Selection box uses K-NN algorithm for classification and the second Feature Selection box uses SVM algorithm with RBF parameter for classification.

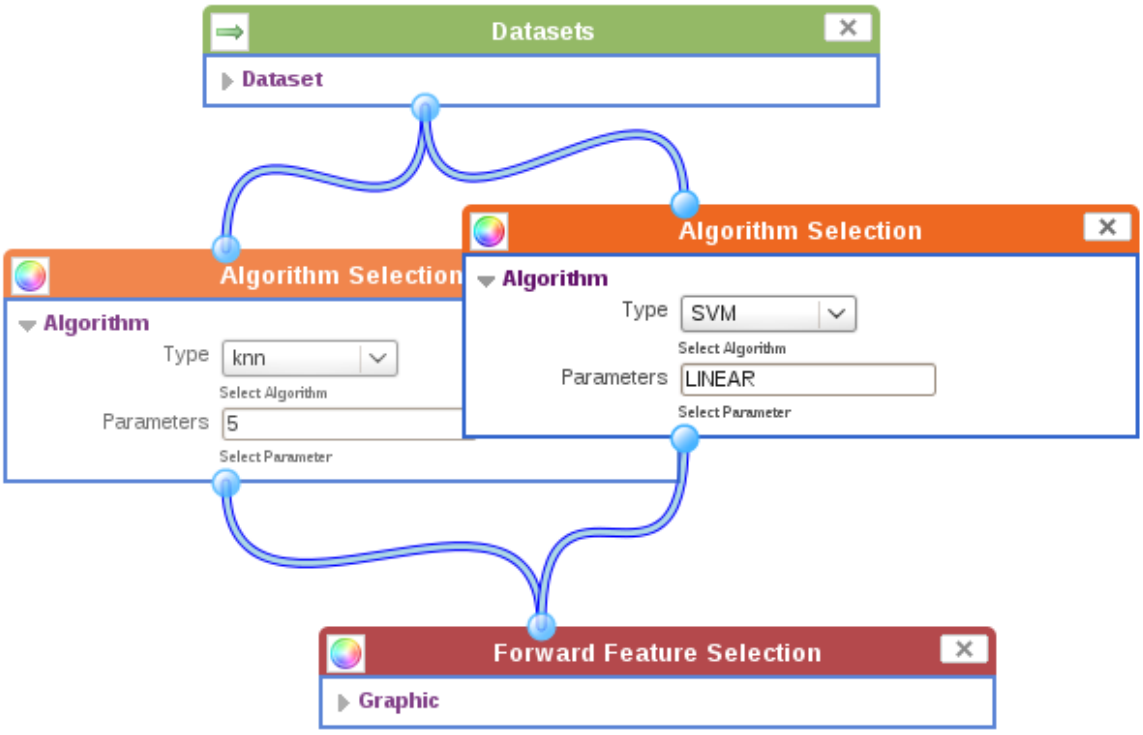

Figure 4.13. Comparison of Feature Selection with K-NN and SVM

Table 4.11 shows the results of the workflow in Figure 4.13. Feature Selection with K-NN stops at the 3rd iteration since any further addition does not decrease the

error. On the other hand, Forward Selection with SVM continues to select features in the 4^{th} iteration rather than stopping because the last addition decreases the error rate.

Table 4.11. Feature Selection on Iris Dataset

	K-NN	**SVM**
Iteration 1	0.9599 (Feature: 4)	0.9599 (Feature: 4)
Iteration 2	0.9666 (Feature: 1)	0.9666 (Feature: 1)
Iteration 3	0.9733 (Feature: 3)	0.9799 (Feature: 3)
Iteration 4	0.9666 (None)	0.98666 (Feature 2)
Selected Features	4, 1, 3	4, 1, 3, 2

4.8. Early Integration

Early integration concatenates inputs as one large vector and then uses it in a single classification algorithm. Figure 4.14 shows how to apply early integration method to a dataset. In this figure, two features of Mfeat dataset (Mfeat Fou and Mfeat Mor) are given to a single classification algorithm. Early Integration concatenates two datasets together and applies 5-NN classifier on this new dataset.

Figure 4.15 shows the confusion matrix after applying the early integration method to Mfeat dataset. Figure 4.16 shows the confusion matrix without applying early integration method. As we can see from the confusion matrices, applying early integration is more accurate, so combining inputs or decisions is useful in increasing accuracy.

4.9. Late Integration

Late integration trains multiple classifiers and combines their decisions by a classification algorithm. Figure 4.17 shows the workflow of Late integration. ML-Lab first classifies the Mfeat dataset with 5-NN and 2-NN, and then combines their decisions using 3-NN algorithm.

Figure 4.14. Workflow of Early Integration Method

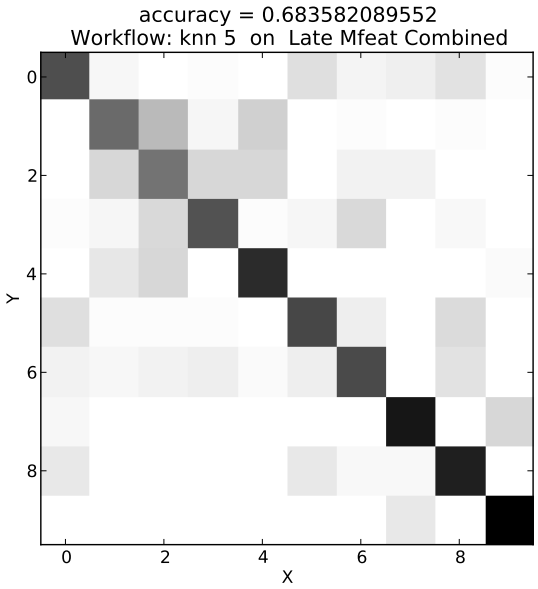

Figure 4.15. Confusion Matrix of Early Integrated Mfeat Dataset

Figure 4.18 shows the confusion matrix after applying late integration method to Mfeat dataset. Figure 4.19 shows the confusion matrix without applying late integration method.

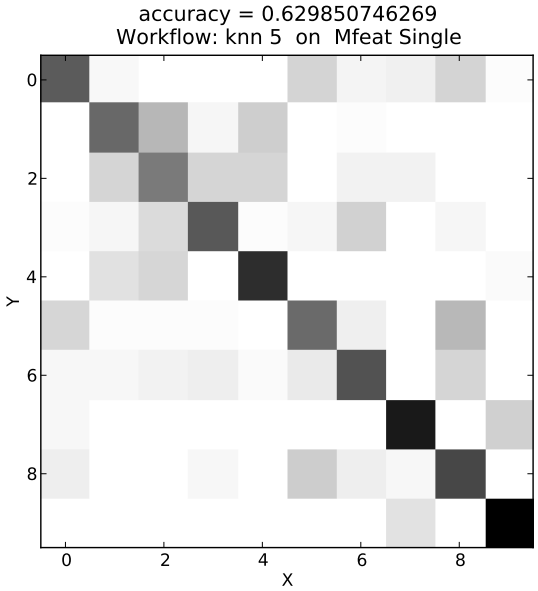

Figure 4.16. 5-NN on single Mfeat dataset

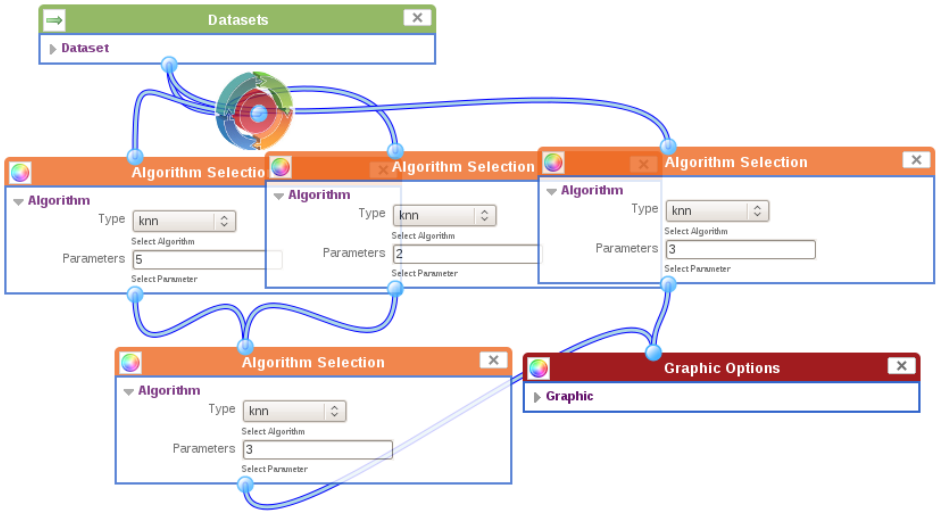

Figure 4.17. Late Integration Method on Mfeat Dataset

We find that there is a small difference between late integration and single k-NN.

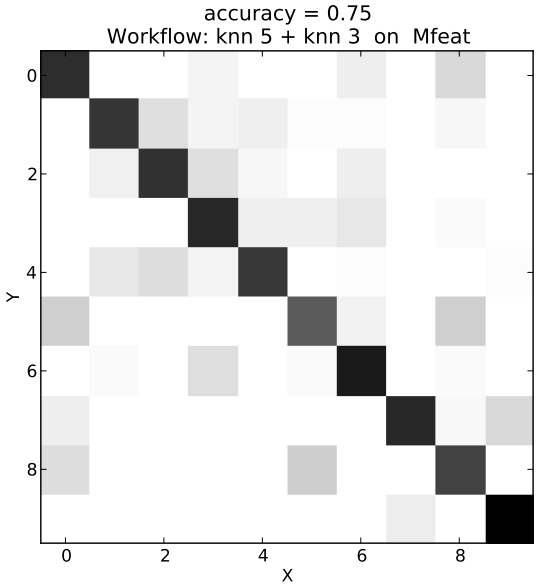

Figure 4.18. 3-NN on single Mfeat dataset

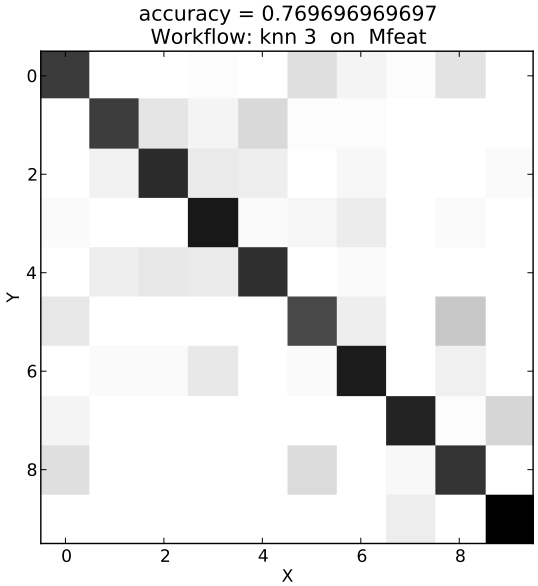

Figure 4.19. Late Integration Method

5. CONCLUSIONS

In this study, an interactive, web-based machine learning framework is developed. Our motivation was to implement such a tool to allow researchers and practitioners compare a wide collection of machine learning algorithms and sophisticated visualization and analysis tools online.

We implemented different classification and dimensionality reduction algorithms, in addition to various resampling, evaluation and integration methods. These methods are tested on three different datasets. We implemented a sequential framework that allows to apply classification after reducing dimensionality, or to apply classification algorithms one by one. From our experiments, we conclude that sequential workflows give better results.

We implemented Early and Late integration methods. From our experiments, we conclude that concatenating as a large vector and then use it in a single classification algorithm (Early method) or training multiple classifiers and combining their decisions by a classification algorithm (Late method) give more accurate results than directly using a single classification algorithm.

In our work, we also implemented resampling and evaluating methods. We implemented K-fold Cross Validation and 5×2 Cross Validation methods for resampling and we proposed Confusion matrices, ROC and PR Curves for evaluating the results better.

As future work, we can focus on running streaming jobs in parallel to make ML-Lab faster. We plan to make both the architecture and the algorithms parallel. We also plan to add new algorithms, especially for clustering.

There are many different data formats (e.g. csv or arff) in use. We plan to implement a data format converter.

APPENDIX A: OPTIONS FOR VISUALIZATION COMPONENT

Color Codes: Users can select one of the colors from the color table. Table A.1 shows the color options for visualization component.

Table A.1. Color table for Visualization Component

Color	**Code**
Blue	b
Green	g
Red	r
Cyan	c
Magenta	m
Yellow	y
Black	k
White	w

Marker: Users can select one of the markers from the marker table. Table A.2 shows the marker options for visualization component.

Title, Xlabel, Ylabel: The title of the graphic, the label of the X axis and the label of the Y axis.

Grid: If clicked, graphics will be generated with a Grid background.

Transparent: If clicked, graphics will be generated with a transparent background.

Landscape Mode: If clicked, graphics will be generated in Landscape mode (horizontal).

Table A.2. Marker table for Visualization Component

Marker	**Description**
.	Point
,	Pixel
o	Circle
s	Square
p	Pentagon
*	Star
+	Plus
x	x
D	Diamond
h	Hexagon
1	Triangle Down
2	Triangle Up
3	Triangle Left
4	Triangle Right

Character Limit: Takes an integer as a parameter and limits the characters to be displayed after the decimal point.

APPENDIX B: ML-Lab Machine Learning Library

ML-Lab's machine learning library supports k-NN, Naive Bayes, C4.5 and SVM classification algorithms and it can be used seperately from ML-Lab. In this appendix, we discuss the functions and usage of this library. **k-NN Classification Algorithm**: ML-Lab's k-NN class has two main functions, **knn.train** and **knn.classify**:

knn.train: Trains a k-NN classifier on a given set. **xs** is a list of observations and **ys** is a list of the class assignments. Thus, **xs** and **ys** should contain the same number of elements. **k** is the number of neighbors that should be examined when doing the classification.

Usage: **ml_lab.knn.train(xs, ys, k)**

Returns: Trained k-NN model.

knn.classify

Classifies an observation into a class by using the trained k-NN model.

Usage: **ml_lab.knn.classify(k-NN, observation)**

Returns: Predicted classes for each dataset element.

k-NN Alternative Input:ML-Lab supports a special parameter format for k-NN algorithm to compare multiple k-NN algorithms easily.

Usage: **First index-Last index, Count** Returns: A list of accuracies (one for each k-NN).

Naive Bayes Classification Algorithm: Naive Bayes method of ML-Lab has two main functions, **naive_bayes.train**,**naive_bayes.classify**:

naive_bayes.train: Trains a Naive Bayes classifier on a given set. **xs** is a list of observations and **ys** is a list of the class assignments. Thus, **xs** and **ys** should contain the same number of elements.

Usage: **ml_lab.naive bayes.train(xs, ys)**

Classifies an observation into a class by using the trained Naive Bayes model.

Usage: **ml_lab.naive_bayes.classify(nb, observation)**

Returns: Predicted classes for each dataset element.

SVM Classification Algorithm: ML-Lab uses LIBSVM [11] which is a public domain software for support vector classification. SVM method of ML-Lab has two main functions, **svm.train**, **svn.classify**:

svm.train: Trains a SVM classifier on a given set. **xs** is a list of observations and **ys** is a list of the class assignments. Thus, **xs** and **ys** should contain the same number of elements.

Usage: **ml_lab.svm.train(xs, ys, parameter)**

Returns: Trained SVM model.

SVM method can take 3 different **parameter**. RBF, LINEAR and SIGMOID kernels: **svm.classify**: Classifies an observation into a class by using the trained SVM model.

Usage: **ml_lab.svm.classify(svm, observation)**

Returns: Predicted classes for each dataset element.

C4.5 Classification Algorithm: C4.5 method of ML-Lab has two main functions, **c45.train**, **c45.classify**.

c45.train: Trains a c45 classifier on a given set. **xs** is a list of observations and **ys** is a list of the class assignments. Thus, **xs** and **ys** should contain the same number of elements.

Usage: **ml_lab.c45.train(xs, ys)**

Returns: Trained C4.5 model.

c45.classify: Classifies an observation into a class by using the trained C4.5 model.

Usage: **ml_lab.c45.classify(c45, observation)**

Returns: Predicted classes for each dataset element.

APPENDIX C: ML-Lab GRAPHICAL LIBRARY

ML-Lab's graphical library supports Confusion Matrix, ROC Curve, PR Curve, 2D/3D Plotting visualization methods. These methods are all implemented in Python and can be used seperately from ML-Lab.

Confusion Matrix function: Takes predicted and actual classes for a dataset and returns the confusion matrix.

Usage: **ml_lab.confmat(predicted, original)**

ROC Curve: Takes the posterior probabilities and original classes for a dataset and returns the ROC curve and AUC values.

Usage: **ml_lab.roc_curve(posterior_probs, original)**

PR Curve: Takes the posterior probabilities and original classes for a dataset and returns the PR curve and AUPC values.

Usage: **ml_lab.pr_curve(posterior_probs, original)**

2D Plotting: Takes the posterior probabilities and original classes for a dataset and returns 2D representation.

Usage: **ml_lab.2D(posterior_probs, original)**

3D Plotting: Takes the posterior probabilities and original classes for a dataset and returns 3D representation.

Usage: **ml_lab.3D(posterior_probs, original)**

REFERENCES

1. Selfridge, O.G., "The Gardens of Learning: A Vision for AI", *Association for the Advancement of Artificial Intelligence Magazine*, Vol. 14, No. 2, 1993.

2. Mitchell, T., *Machine Learning*, McGraw Hill, 1997.

3. Kuncheva, L.I., *Combining pattern classifiers: methods and algorithm*, Wiley-Interscience, 2004.

4. Hall, M., E.Frank, G.Holmes, B.Pfahringer, P.Reutemann, and I.H. Witten, "The WEKA Data Mining Software: An Update", *SIGKDD Explorations*, Vol. 11, Issue 1, 2009.

5. Demsar, J. and B.Zupan, "From Experimental Machine Learning to Interactive Data Mining, White Paper", software available at `http://www.ailab.si/orange`, 2010.

6. Alpaydin, E., *Introduction To Machine Learning*, The MIT Press, 2004.

7. Berry, M. and M.Browne, *Lecture Notes in Data Mining*, World Scientific Publishing, 2006.

8. Olson, D.L. and D.Delen, *Advanced Data Mining Techniques*, Springer, 2008.

9. Marsland, S., *Machine Learning: An Algoritmic Perspective*, Chapman and Hall/CRC, 2009.

10. Burges, C.J., *A Tutorial on Support Vector Machines for Pattern Recognition*, Vol. 2, No. 2, Springer Netherlands, 1998.

11. Chang, C.-C. and C.-J. Lin, *LIBSVM: a library for support vector machines*, software available at `http://www.csie.ntu.edu.tw/cjlin/libsvm`, 2001.

12. Quinlan, J.R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993.

13. Dunham, M., *Data Mining: Introductory and Advanced Topics*, Prentice Hall, 2003.

14. Ruggeiri, S., "Efficient C4.5", 2002.

15. Larose, D.T., *Discovering Knowledge in Data: An Introduction to Data Mining*, John Wiley and Sons, 2005.

16. Tenenbaum, J.B., V.Silva, and J.C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction", 2000.

17. Dijkstra, E.W., "A note on two problems in connexion with graphs", 1959.

18. Borgwardt, K., "Data Mining in Bioinformatics", *Bioinformatics Group MPIs Tubingen*, 2000.

19. Fawcett, T., "An introduction to ROC analysis", *Pattern Recognition Letters*, Vol.27, 2006.

20. Noble, W.S., "Support vector machine applications in computational biology", *In Kernel Methods in Computational Biology*, 2004.

21. Django, "Django, a High-level Python Web framework", software available at `http://www.djangoproject.com`, 2010.

22. Wireit, "A Javascript Wiring Library", software available at `http://javascript.neyric.com/wireit`, 2010.

23. Ascher, D., "Numerical Python", software available at `http://numpy.org`, 2001.

24. Barrett, P., J.Hunter, and P.Greenfield, "Matplotlib - A Portable Python Plotting

Package", *Astronomical Data Analysis Software and Systems XIV*, Vol. 347, 2004.