University of the Philippines – Manila

College of Arts and Sciences

Department of Physical Sciences and Mathematics

Lung Cancer Classification Tool Using Microarray Data and Support Vector Machines

A Special Problem in Partial Fulfillment

Of the Requirements for the Degree of

Bachelor of Science in Computer Science

Jennifer P. Cabrera

April 2014

ACCEPTANCE SHEET

The Special Problem entitled “Lung Cancer Classification Tool Using Microarray Data and Support Vector Machines” prepared and submitted by Jennifer P. Cabrera in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

_____

Geoffrey A. Solano, M.Sc.

Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (candidate)	_____	_____
2. Avegail D. Carpio, M.Sc.	_____	_____
3. Richard Bryann L. Chua, M. Sc.	_____	_____
4. Aldrich Colin K. Co, M S. (candidate)	_____	_____
5. Ma. Sheila A. Magboo, M.Sc.	_____	_____
6. Vincent Peter C. Magboo, M.D, M.Sc.	_____	_____
7. Bernie B. Terrado, M.Sc, (candidate)	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

_____

Ma. Sheila A. Magboo, M.Sc.

Unit Head

Mathematical and Computing Sciences Unit

Department of Physical Sciences and Mathematics

_____

Marcelina B. Lirazan, Ph.D.

Chair

Department of Physical Sciences Mathematics

_____

Alex C. Gonzaga, Ph.D., Dr.Eng.

Dean

College of Arts and Sciences

Abstract

Lung cancer is one of the deadliest types of cancer in country and around the world. Epidemiologic studies have shown that genetic variability is among the factors that affect a person's susceptibility to lung cancer. This study proposes a system that will utilize gene expression data to predict the presence or absence of lung cancer, predict the specific type of lung cancer should it be present, and determine marker genes that are attributable to the specific kind of the disease. The proposed system would help in the faster diagnosis and serve as a reliable adjunct approach to current lung cancer classification methods.

Keywords: lung cancer diagnosis, lung cancer classification, support vector machines, microarray gene expression data, preprocessing and feature selection techniques

CONTENTS

I. INTRODUCTION

A. Background of the Study

Lung cancer is the result of the uncontrolled growth of abnormal cells in one or both lungs. These abnormal cells do not die and instead divide rapidly and form tumors that can either be benign or malignant [1, 2]. Cancer refers to the malignant tumors as they grow aggressively, disrupt the normal function of the lungs, and spread to other parts of the body [2].

Lung cancer is one of the deadliest types of cancer, accounting for more than 3,000 deaths everyday worldwide, and more deaths than breast, colon, kidney, liver, skin and prostate cancers combined [4]. According to the World Health Organization (WHO), lung cancer is the top cause of cancer related deaths in the Philippines. It is the most common cancer type in men and third most common in women in 2010 [3].

Lung cancer can be classified into two main types: small cell lung cancer (SCLC) and non small cell lung cancer (NSCLC). SCLC is an aggressive type of lung cancer caused primarily by smoking and is responsible for 15 percent of all diagnosed cases. NSCLC is the most common type of lung cancer, with various causes including smoking. NSCLC can further be classified into four different types: namely squamous cell carcinoma, adenocarcinoma, bronchoalveolar carcinoma, and large cell undifferentiated carcinoma [4].

Lung cancer is not common before the 1930's and only rose dramatically as a result of increased tobacco smoking. The incidence of the deadly disease is strongly correlated with cigarette smoking as well as passive smoking. Other causes are exposure to asbestos fibers, radon gas, presence of certain lung diseases (e.g. chronic obstructive pulmonary disease (COPD)), prior history of lung cancer, and air pollution [2].

Because of the fact that not all smokers eventually develop lung cancer, it is suggested that cancer can be a result of individual genetic disposition inherited from family members [1, 2]. This means that some people are born with certain genetic mutations or faults in a gene(s) that make them statistically more susceptible to the development of lung cancer later in life. Many studies have shown that smoking or nonsmoking relatives of those who had lung cancer are more at risk of having the disease than the general population [2]. Epidemiologic studies provide evidence that genetic variability in the individual response to carcinogens may modify the susceptibility to lung cancer. Among this is a very recent study conducted by a team of researchers from the United States National Cancer Institute among 14,000 Asian women and found out that Asian women, whether smokers or not, are more prone to developing cancer due to their genetic variations [5].

Currently, information technology plays a major role in many fields, including medicine. Many studies and techniques have combined information technology techniques such as machine learning algorithms and statistical methods in the identification and classification of diseases, including cancer.

B. Statement of the Problem

Currently, lung cancer classification is based on clinicopathological features only. Doctors need to run several examinations like X-Ray, CT scan, and biopsy in order to detect the presence of lung cancer in the patient. Also, aside from tests for detection, further examinations like PET scans and MRI are also needed to identify the type of lung cancer present [61]. This study aims to make this process faster and easier but at the same time reliable through the use of DNA microarray technology. The results of this process can also be used in the validation of examinations and support diagnosis.

Since DNA microarray technology is now becoming widely used, data gathered from this technology can also be used to aid in the effective, reliable and fast identification and classification of lung cancer patients. This study aims to create a machine learning classification system that would analyze gene expression data from oligonucleotide microarrays to determine whether a patient has lung cancer or not, and use Support Vector Machines (SVM) to classify the kind of lung cancer present should the patient be affirmative of having it.

More specifically, this study seeks to answer the following questions:

i. How can gene expression data from oligonucleotide microarrays be used in the prediction of the lung condition of the patient?
ii. How the genes that are most are contribute to the lung condition selected?
iii. How are the genes to be used in the prediction of the type of lung cancer present in the patient selected?
iv. How are genes most associated to a particular lung condition identified?
v. How are the results of the analysis generated?

C. Objectives of the Study

The study aims to develop an automated diagnostic system that takes oligonucleotide microarray expression data as input. Using statistical methods and information learned during the system "training", the system predicts the presence or absence of lung cancer and its corresponding type(s) should it be present.

More specifically, the study has the following objectives:

i. To "train" the system so that it has the capability to classify whether a patient has a normal lung or has lung cancer and which of the main classes is present.
ii. To "train" the system to that it has the capability to subclassify adenocarcima (ADCA)-positive data into its corresponding ADCA subclass.
iii. To preprocess data so that it is free from non-biological factors that may affect the accuracy of the prediction of the system.
iv. To select the genes (features) that are most likely to contribute in the identification of the lung condition.
v. To identify the gene markers for a specific lung condition.
vi. To predict the presence or absence of lung cancer on test expression data (Normal lung and Lung Cancer Positive).

vii. To classify lung cancer positive expression data into its corresponding lung cancer class (small cell lung carcinoma (SCLC), squamous cell lung carcinoma (SQ), pulmonary carcinoids (COID), and adenocarcinoma (ADCA)).

viii. To classify ADCA-positive expression data into its corresponding ADCA class (subclasses C1, C2, C3, C4).

ix. To provide classification result based on features selected from the microarray data analysis.

D. Significance of the Study

This study proves to be able to assist in the fast and reliable diagnosis of the presence or absence of lung cancer, as well as in the identification of the specific kind of lung cancer should it be present on the patient's gene expression data. It also helps in the prediction of specific ADCA subclasses that might be present in the patient.

A simultaneous analysis of gene expression data can be a powerful adjunct approach in the diagnosis of lung cancer. The results generated by such a system can be used to support oncological test findings as well as identify a patient's susceptibility to the disease through analysis findings of his/her gene data. Marker genes identified by the system serve to validate existing knowledge about genes associated with the disease.

This study can also help in the informed selection of the appropriate treatment or therapy for a patient based on the marker genes that the system identifies. It also aids in the identification of novel molecular targets for chemotherapy and drug design. And finally, the success of this system could mean lengthening or saving the lives of lung cancer patients.

E. Scope and Limitations

The following are the scope and limitations of the proposed system:

i. The system enables the user to choose preprocessing method(s) that the system will use for the data. If no method is chosen, normalization by decimal scaling is designated as the default preprocessing method.

ii. The system enables the user to choose the method the system will use to select genes more associated with the lung condition. If no method is chosen, the standard deviation method is the alternative method.

iii. The system enables the user to specify the number of genes to be selected.

iv. The system identifies the marker genes for a specific lung condition based only on the selected genes attributable to the lung condition.

v. The system predicts the presence or absence of lung cancer based from the classification rule generated after training the data.

vi. If lung cancer is predicted, the system identifies the specific type of lung cancer present in the given data.

F. Assumptions

The following parameters are assumed for the effective run of the proposed system:

i. Input dataset is already extracted and in numerical format.

ii. Oligonucleotide microarray data is considered for input datasets.

iii. Training data and testing data follows the desired format (.csv). The rows correspond to the gene expression data and the columns correspond to the sample/experiment/patient.

iv. It is assumed that the training datasets and testing datasets do not have missing expression values.

v. The system is designated for use of lung specialists/doctor, molecular biologists/biotechnologists, lung cancer specialists, and other health care professionals.

II. REVIEW OF RELATED LITERATURE

Over the years, DNA microarrays have made possible the creation of large datasets of molecular information characterizing complex biological systems. Various machine learning algorithms have resulted in many applications in statistics and medicine, especially in the classification of various diseases. In many research undertakings, several classifiers and data mining models have been used that targeted the appropriate categorization of cancer and other diseases using oligonucleotide and cDNA microarray data.

The following are related literature and studies that used concepts, methods and algorithms that will be used in the proposed system over the past years.

A. Classification/Prediction Using Microarray Gene Expression Data

Microarray analysis of gene expression data makes it possible to search systematically for molecular markers of cancer classification and outcome prediction. In 2001, Bhattacharjee et al analyzed mRNA expression data using oligonucleotide microarrays to classify human lung carcinomas [6]. Using hierarchical clustering on normalized data, they discovered the different types of lung cancer, including subtypes of ADCA, the type of lung cancer that proves to be challenging to sub-classify. Aside from clustering the different types, the study also identified the marker genes the best defined the different clusters, as well as compared the survival within the clusters using Kaplan-Meier (K-M) curves.

A similar study was also conducted by Garber et al using cDNA microarrays, recapitulated the conventional division of lung tumors and identified ADCA subtypes [7]. The study also provided a detailed analysis and comparison of ADCA subgroup properties based on the gene expression characteristics.

Related studies have proven microarray gene expression data analysis to be useful in the detection of mutations of colorectal cancer [9], and the prediction of courses of neuroblastoma [10]. Supervised prediction analysis for microarrays (PAM) yielded a cross-validated accuracy of 99% in the latter study. Other related studies analyzed breast cancer gene expression profiles [15].

In 2012, Ramani and Jacob designed a computational strategy to predict the class of lung tumors from structural and physicochemical properties of protein sequences obtained from the genes defined by microarray analysis [9]. Using hybrid feature selection techniques and Bayesian Network prediction, they generated a cross validation accuracy of 87.6%.

Other methods and techniques proposed for classification using microarray data are Naïve Bayesian Classifier [16], gene pairs [17], hyper-box enclosure (HBE) [18], meta-analysis techniques [19], locally linear discriminant embedding [20], and single genes [21] applied to various datasets such as leukemia, prostate cancer, breast cancer, lymphoma, and lung cancer.

B. Support Vector Machines

Support Vector Machines (SVMs) are supervised learning models with associated learning algorithms that analyze example data and recognize patterns in it [11]. Given a training set, an SVM training algorithm finds a decision rule which explains the training set well and can be used to assign labels to new unlabeled examples. SVM is a model used for classification and regression. This particular machine learning algorithm has shown great promise in a variety of biological classification tasks, including gene expression microarrays [12].

In 2010, Gomes et al used SVM to classify dengue fever patients into dengue fever (DF) and dengue haemorrhagic fever (DHF) based from cDNA microarray data [13]. Results revealed that there are seven genes central in differentiating DF patients from DHF patients. Also, an overall prediction accuracy of ~96% was rendered using the algorithm.

Another excellent example of the use of SVM in microarray gene expression data classification is Sanchez-Carbayos' application to classify and stratify bladder tumors on the basis of stage, node metastases, and overall survival using oligonucleotide microarray gene expression data [14]. Hierarchical clustering classified the different bladder cancer subtypes and the SVM predictive algorithm was used for tumor staging, rendering an 89% accuracy rate.

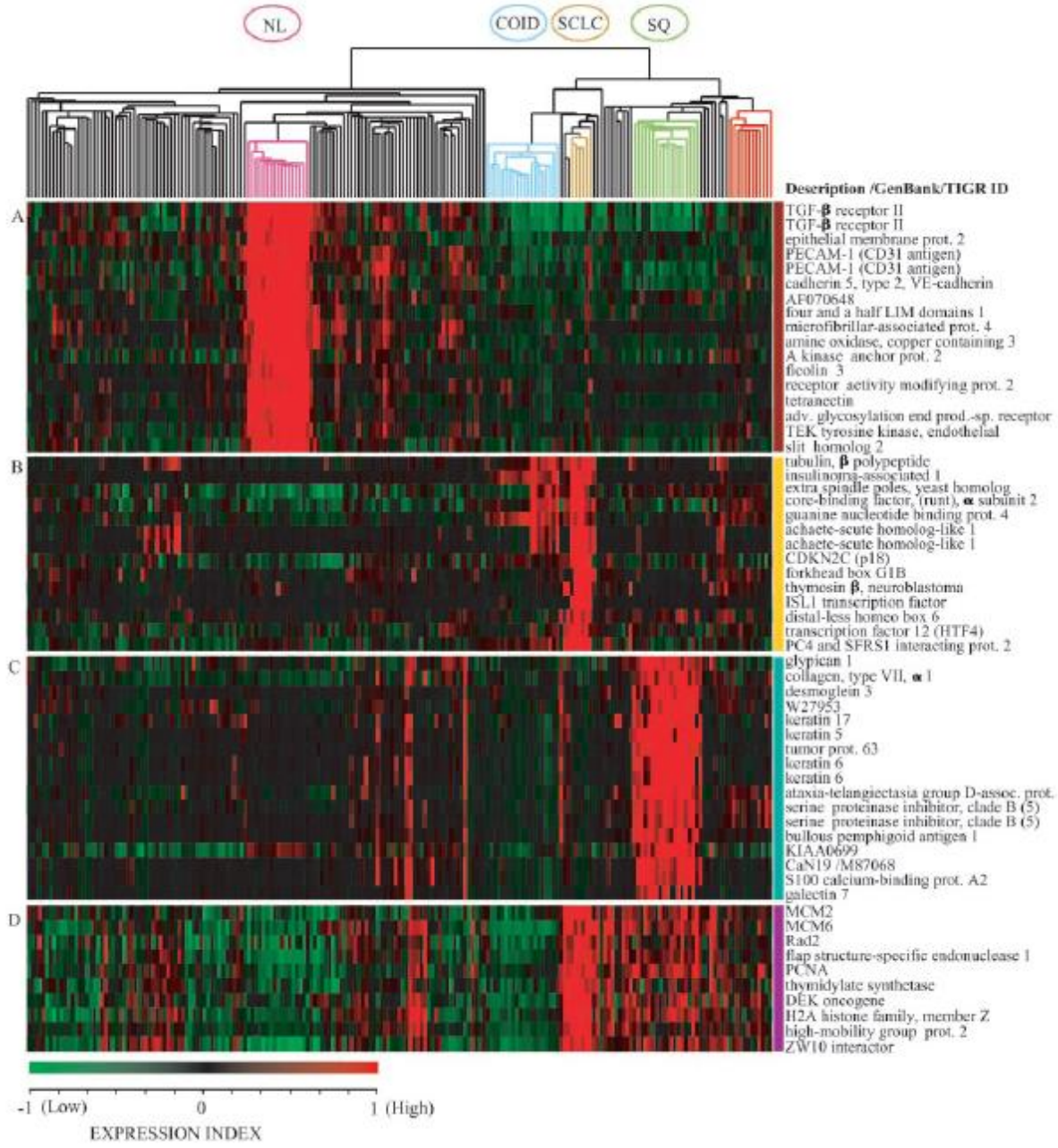


Figure 1: Hierarchical clustering results defining classes and subclasses of lung tumor [6]

A variant of SVM, the bagging support vector machines (bSVM), was applied in the classification and diagnosis of leukemia based on gene expression data [22]. bSVM trains each SVM separately using bootstrap technique and then aggregates the performances of each SVM by majority voting. An accuracy of 87.5%-92.5% was rendered, and outperformed other classification methods.

SVM is originally designed for binary classification of data. But now, it has been extended to accommodate classification of K classes. The most common SVM multiclass approaches are one-vs-one (OVO) and one-vs-all (OVA) [24]. In the OVA, each class is learned against the other classes, resulting in k classifiers. In OVO, on the other hand, builds $k(k-1)/2$ classifiers, with each one learning the separation between two classes.

An application of multiclass SVM classification was used in fingerprint classification using the OVA scheme and dynamically ordered with naïve Bayes classifiers [23]. An accuracy of 90.8% for five-class classification was achieved. On the other hand, Yu et al performed a confidence evaluation of the OVO and OVA approaches and found that OVA SVM is superior to OVO because it yielded higher predictive accuracy in majority of the datasets used [25].

C. Data Normalization and Preprocessing

Preprocessing and normalization techniques are needed to accurately interpret the results by removing aberrations and outliers in the data. In microarray data, this is important in order to remove non-biological factors such as preparation, transcription, amplification, abundances, labeling differences, hybridization parameters, scanner differences, etc [26].

Hoffmann et al demonstrated the effect of normalization on the detection of differentially expressed genes [34]. Using four different normalization methods and three statistical algorithms for detecting differentially expressed genes, it was found that normalization has profound influence in the result. They found that a high concordance between different methods can only be achieved by using the same normalization procedure.

For oligonucleotide microarrays, Gomez mentions scale normalization and quantile normalization techniques as the most common normalization methods [26]. Other suggested normalization techniques are min-max normalization, normalization by decimal scaling [27] and z-score transformation [27, 30].

Bolstad et al presented quantile normalization as a complete data method because it makes us of data from all arrays in an experiment to form the normalization relation. They also compared it other methods that use a baseline array, including scaling normalization. Results showed that quantile normalization reduced the variation of a probeset measure across a set of arrays to a greater degree than the scaling method and unnormalized data. In relation to bias and speed, the quantile method also had an advantage over the other methods. For scale normalization, the problem of choosing the right baseline was discussed, as the quality of the results depends on the baseline. The paper concludes that the quantile method should be used in preference to other methods [28].

Cheadle et al on the other hand utilized z-score transformation as a preprocessing method for oligonucleotide microarray data [30]. Z-scores have been successfully used directly in hierarchical clustering, k-means clustering, self-organizing maps (SOM), principal component analysis (PCA), and multidimensional scaling. Z-scores provide a relative, semi-quantitative estimate of gene expression levels, and, as such, form the basis of comparison of hybridization intensity data among many experiments within the same array type.

D. Gene Selection and Marker Gene Identification

Gene selection is a process of selecting the most informative genes which are predictive to its related class for classification. Hu et al explained that the most popular filter gene selection methods, signal-to-noise ratio (SNR), and correlation coefficient, are called *ranking* and has been applied to cancer classification. Filter methods are popular as these methods can reduce the dataset size before classification. Hu et al have also shown that SVM improves its prediction accuracy by to 15%, with SNR as the most effective method. The results indicate that gene selection methods in general improve the prediction accuracy of classification. It also shows that the number of selected genes affects the classification accuracy. The overall performance is better when datasets contain 50 or 100 genes. Finally, their study shows that not all gene selection methods improve the performance of classification. If we select SVMs as the classification algorithm, the correlation coefficient or SNR gene selection methods are

better for processing [29]. SNR has been used in cancer classification of lung cancer and leukemia data [6, 31, 32].

Another gene selection method, median absolute deviation (MAD), was used in the classification mediastinal Large B-Cell Lymphoma [33].

Other methods for detecting differentially expressed genes were F-test for parametric ANOVA, and Kruskal-Wallis test for nonparametric ANOVA [34]. These methods address the problem of gene selection for multiclass experiments containing more than two experimental conditions. Here, differential gene expression is detected by comparing variances within experimental conditions to variances across experimental conditions. F-test however relies on certain assumptions, namely that the underlying data are normally distributed with equal variances across experimental conditions. These assumptions must not necessarily be met and analysis of datasets show that they are often not fulfilled. Aside from Kruskal-Wallis test for nonparametric ANOVA [59], Saeys et al also listed several other model free feature selection methods in the microarray domain, namely Wilcoxon rank sum and between-with classes sum of squares (BSS/WSS) methods, as more appropriate to use [58].

Other mentioned feature selection methods are the chi-square statistic, Euclidian distance, t-test, and information gain (IG) [57].

III. THEORETICAL FRAMEWORK

DEFINITION OF TERMS

A. **Carcinoma** – cancer

B. **Lung Cancer/Lung Carcinoma** – results from the uncontrolled growth of abnormal cells that begin in one or both lungs that do not die and divide rapidly to form tumors that can either be benign or malignant [1,2]. It has two main types:

   a. **Small Cell Lung Carcinoma (SCLC)** – the type of lung cancer that spreads faster than NSCLC. It accounts for about 10-15 percent of all lung cancer cases [4].

   b. **Non Small Cell Lung Carcinoma (NSCLC)** – type of lung cancer that accounts for the remaining 85 percent of all cases. It is the most common type and is mainly caused by smoking [4]. It can be further categorized into three subclasses:

      i. **Adenocarcinoma (ADCA)** – type of NSCLC that accounts for about 50 percent of all lung cases. It usually begins in tissues as the outer parts of the lung and may already be present for a long time before symptoms appear and diagnosed. It is often found in nonsmokers. It is also the most common type of cancer in women, in people under 45, and among all Asians [35].

      Studies show the ADCA has four distinct subclasses discovered through classification of mRNA expression data [6, 7].

      ii. **Squamous Cell Carcinoma (SQ)** – type of NSCLC that is the second most common type of lung cancer, making up for about 25-30 percent of all cases. It usually occurs in the central parts of the lung or in the main airway branches. It is almost always caused by smoking history. Secondary causes are age, family history, secondhand smoke, mineral dust, asbestos, or radon. Its symptoms develop slowly [36].

      iii. **Pulmonary Carcinoids (COID)** – It is a rare tumor that originates from the cell lining the airways. It accounts for only 1-2 percent of all cases and comes from neuroendocrine cells. Smoking is not a major cause. It has a relatively higher cure and survival rates [37].

C. **Gene** – basic physical unit of heredity. They are made up of DNA and encode instructions to make protein molecules. It determines what the organism will be like, its appearance and how it will behave in the environment. Genes are now widely used to test a person's susceptibility to cancer and other diseases [38-40].

D. **Gene Expression** – process by which gene information in the DNA is transcribed into messenger RNA (mRNA) molecules and is translated into proteins. Disruption in gene expression is responsible for many diseases [41].

E. **Oligonucleotide** – or oligo, is a short fragment of a single-stranded DNA that is typically 5 to 50 nucleotides long [41].

F. **Deoxyribonucleic Acid (DNA)** – hereditary material found in the nucleus of cells and composes genes [42].

G. **Metastasis** – process of spread of tumor to distant areas of the body [2].

H. **Biomarker** – or biological marker, is a substance that is used as an indicator of a biological state [43].

MICROARRAY

A microarray is a tool for analyzing gene expression. It consists of a small membrane of glass slide that contains samples of many genes arranged in a regular pattern [41]. It utilizes the selective nature of DNA-DNA or DNA-RNA hybridization [44]. Applications of microarrays include comparison of the different tissues, effects of drug treatment, cancer development, identification of organisms, timepoint, etc.

DNA Microarray

DNA microarrays are small solid supports onto which sequences of genes are immobilized at fixed locations (probes) in an orderly or fixed arrangement. The spot in the array can be DNA, cDNA or oligonucleotides. Their locations are used to identify a particular gene sequence [41].

In a DNA microarray experiment, there are two different types of cell or tissue: healthy and diseased, both containing an identical set of genes (target DNA). The target DNA is placed in the microarray. To determine the gene expression profile of each cell or tissue, mRNA is isolated from each cell type (probe mRNA) and used as a template to generate corresponding DNA, cDNA or oligonucleotides (with fluorescent tags attached to them). Then, the target DNA and probe mRNA are mixed and allowed to hybridize (complementary probe mRNA will bind to target DNA sites in the array corresponding to the genes expressed in each cell) [41, 45].

Oligonucleotide Microarrays

Oligonucleotide microarrays use DNA oligo probes and they are synthesized directly on the microarray through combinatorial chemistry and photolithography. They use commercial microarrays (Affymetrix, Nimblegen and Combimatrix) and are often expensive [44, 46].

Oligonucleotide arrays consist of short oligonucleotides rather than that of much longer cDNA molecules. As a result, every gene to be examined is represented by several probes known as probe set. Unlike cDNA microarrays, oligonucleotide microarrays use one-color detection. It determines the gene expression level for each sample. It uses one array per sample [44]. The probe cells in oligonucleotide microarrays are square-shaped features on the chip containing millions of copies of a single 25-mer probe. The intensity of the pixels of the resulting microarray image is combined to form one number representing the expression level for the probe cell oligonucleotide [46].

MACHINE LEARNING

Machine learning is a branch of Artificial Intelligence (AI) that gives computers the capability to learn without being explicitly programmed. It deals with the design of computer systems that take advantage of sample data and past experiences to improve its own performance in a specific task of set of tasks [46-47]. It uses the theories of statistics to build mathematical models that seek to detect patterns in the data and make inferences from it [48].

Supervised Machine Learning

Supervised machine learning is the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. Its goal is to build a concise model of the distribution of class labels in terms of predictor features. The resulting classifier is then used to assign labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown.

CLASSIFICATION (STATISTICAL CLASSIFICATION)

In machine learning, classification, or statistical classification is the problem of building discriminative models to separate and classify new data points [49]. It is considered a case of supervised learning because the system, or classifier, needs to be first fed a set of already classified data, or training set, to which it would learn the characteristics of the labels to create models to process future unlabeled input [50]. Classification can have a discrete or categorical random variable of interest. It also involves a fixed set of categories [51].

To build a classifier, we need to accomplish the following steps [49].

1. Define classes or attributes.

   These classes or attributes are the labels in which we want to classify our data as. They could be stated using explicit rules or better defined through the training examples in the training set. The **training set** is a set of already classified data that is fed into the classifier for it to use to "learn". We refer to this as the set $X = \{x_1, x_2, \cdots, x_N\}$.

2. Feature extraction and definition of the feature space.

   From the training data we can determine the variables, or features, that we want to measure to assess which class or attribute a future unlabeled sample belongs. The features can either

be discrete or continuous. If m features are selected, each sample contains an $m \times 1$ row vector x_i, for $i \in \{1, \cdots, N\}$of data, and we refer to the space $\mathbb{R}^m$ as the **feature space**. The training set is an $N \times m$ matrix, where x_{ij} represents the j^{th} feature of the i^{th} sample.

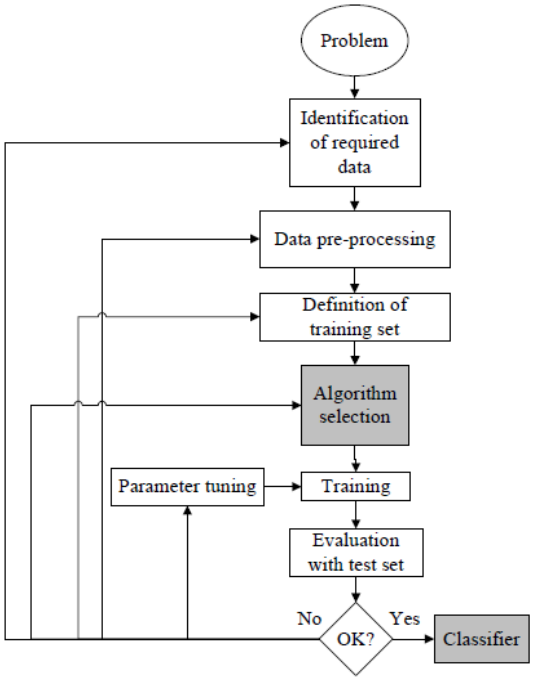


Figure 2. The process of supervised machine learning [52]

3. Define the decision function or algorithm.

   Using the training data X, the classifier will build a decision function $D(x)$ that makes a new data point and produces a prediction of its population.

4. Measure the performance.

   Performance or accuracy of the classifier is calculated according to the (weighted) error rate.

Classification has two types: binary classification and multiclass classification. Binary classification involves only two classes while multiclass involves three or more classes.

DATA NORMALIZATION AND PREPROCESSING

Data normalization and preprocessing must be done prior to gene expression analysis to obtain biologically meaningful measurements. Normalization techniques are needed to accurately interpret the results by removing aberrations and outliers due to non-biological factors (preparation, transcription, amplification, abundances, labeling differences, hybridization parameters, scanner differences, image analysis, etc) and revealing the patterns and outliers of actual biological factors (diseased vs. normal) [53]. Its goal is to manipulate data to make measurements from multiple arrays comparable [53-54]. Preprocessing, on the other hand, involves scale linearization, summarization, and missing value imputation.

Several normalization methods for oligonucleotide microarray data have already been proposed. Among these methods are the following:

Normalization by Decimal Scaling

Normalization by decimal scaling normalizes data by moving the decimal point of values of an attribute A. The number of decimal points moved depends on the maximum absolute value of A [55]. A value v_i, of A is normalized to v'_i by computing

$$v'_i = \frac{v_i}{10^j}$$

Where j is the smallest integer such that $\max(|v'_i|) < 1$.

Quantile Normalization

Quantile normalization method seeks to impose the same empirical distribution of intensities across arrays and across channels by transforming the distribution of intensities from one distribution to another [26, 28]. A quantile-quantile plot will have a straight diagonal line, with slope 1 and intercept 0, if two data vectors have the same distribution. Thus plotting the quantiles of two data vectors against each

other and then projecting each point of a 45-degree diagonal line yields a transformation that gives the same distribution to both data vectors.

Given n vectors of length p, form X of dimension $p \times n$, where each array is a column.

Sort each column of X separately to give X_s.

Take the mean, across rows, of X_s and create X'_s, an array of the same dimension as X, but where all values in each row are equal to the row means of X_s.

Get X_n by arranging each column of X'_s to have the same ordering as the corresponding input vector.

Algorithm 1. Quantile Normalization Algorithm

Min-Max Normalization

Min-max normalization performs a linear transformation on the original data [55]. Suppose that $A_{min}$and $A_{max}$are the minimum and maximum values of an attribute A. Min-max normalization maps a value, v_i, of A to v'_iin the range $[A_{new\_min}, A_{new\_max}]$ by computing

$$v'_i = \left(\frac{v_i - A_{min}}{A_{max} - A_{min}}\right) * (A_{new\_max} - A_{new\_min}) + A_{new\_min}$$

Min-max normalization preserves the relationships among the original data values. It will encounter an “out-of-bounds” error if a future input case for normalization falls outside of the original data range for A.

Z-Score Transformation

Z-score transformation is a common normalization procedure for spotted cDNA arrays [30]. It is a method of standardizing data across a wide range of experiments and allows the comparison of microarray data independent of the original hybridization intensities. Cheadle et al. has proven that z-

score transformation can be used in Affymetrix oligonucleotide microarray data without significant loss of information content [30].

Z-scores are proportional to the intensity of the original hybridization signal. The value of the z-score is directly reflective of the underlying hybridization values (i.e., higher positive z-scores represent the most highly expressed genes; lower negative z-scores represent the least expressed genes). The z-scores provide a useful and intuitive method for visualizing and interpreting very large amounts of data in their natural biological context.

In z-score normalization (or zero-mean normalization), the values for an attribute, A, are normalized based on the mean (i.e., average) and standard deviation A [55]. A value, v_i, of $A$is normalized to $v'_i$by computing

$$v'_i = \frac{v_i - \overline{A}}{\sigma_A}$$

Where $\overline{A}$ and $\sigma_A$are the mean and standard deviation, respectively, of attribute A. This method of normalization is useful when the actual minimum and maximum of attribute A are unknown, or when these are outliers that dominate the min-max normalization.

GENE SELECTION AND MARKER GENE IDENTIFICATION

Feature selection is important because we need to know which genes are most relevant to the classification task. Removing noise or irrelevant genes might improve the performance of the classifier. If we have a number of independent features that correlate well with the classes, the machine learning process is easier and more accurate. Also, a candidate list of important genes can be used to further understand the biology of the diseases and can be used in the development of specific treatments targeting these molecular biomarkers.

Median Absolute Deviation (MAD)

Median Absolute Deviation can be used to rank genes according to a variation filter so as to give higher priority to genes with expression values showing maximal variation across the samples being analyzed [33].

$$\mathrm{MAD}(g) = \frac{1}{N}\sum_{i-1}^{N} |g_i - \mathrm{med}(g)|$$

where N is the number of samples, g_i denotes g's expression level in sample i, and $\mathrm{med}(g)$ denotes the median expression level of gene g.

Signal-to-Noise Ratio (SNR)

The signal-to-noise ratio statistic is a filter feature selection method proposed by Golub et al. in 1999. Filter feature selection methods select features according to criteria that are independent of those criteria that the classifier optimizes [56].

For a gene j, the SNR statistic is computed by:

$$S(j) = \frac{\mu_+(j) - \mu_-(j)}{\sigma_+(j) + \sigma_-(j)} \qquad (3)$$

where $\mu_+(j)$ and $\mu_-(j)$ are the means of the classes $+1$ and -1 for the jth gene. Similarly, $\sigma_+(j)$ and $\sigma_-(j)$ are the standard deviations for the two classes for the jth gene. Genes that give the most positive values are most correlated with class $+1$, and genes that give the most negative values are most correlated with class -1.

SUPPORT VECTOR MACHINES (SVM)

Support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze example data and recognize patterns in it [60]. It is a model used for classification and regression. Given a training set, an SVM training algorithm finds a decision rule which explains the training set well and can be used to assign labels to new unlabeled examples [61-62]. It aims to find a decision rule with high generalization ability.

Training the System:

For linearly separable data, a separating hyperplane is defined by

$$\text{hyperplane} = (v \cdot x) + b = 0 \quad (\mathbf{4})$$

where w is the normal vector, b is the offset, and the operation $\langle .,. \rangle$ is called scalar product or dot product. Training is the process of choosing w and b from the labeled examples in the training set [62]. SVM maximizes the margin or distance around the separating hyperplane. The margin serves as a measure of confidence as we intuitively want to separate the two classes as much as possible [62-63]. The points nearest to the separating hyperplane are called the Support Vectors. They determine the position of the hyperplane. Mathematically, the weighted sum of the Support Vectors is the normal vector of the hyperplane.

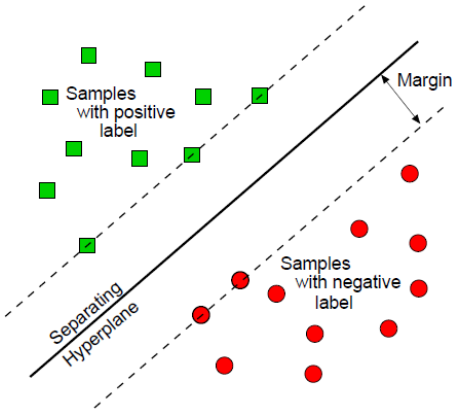


Figure 3. Visualization of Maximal Margin Hyperplane [62]

To find the maximum margin hyperplane, say we have the training set $\{x_i, y_i\}$ where $x_i \in \mathbb{R}^n$, $y_i \in \{-1, +1\}$. We find the hyperplane by (4) with $\|v\| = 1$. The margin δ is

$$\delta = y(v \cdot x) \quad (\mathbf{5})$$

To maximize δ, we subject it to $y_i(v \cdot x_i) \geq \delta$ and $\|v\| = 1$. We set $w = \frac{v}{\delta} \Rightarrow \|w\| = \frac{1}{\delta}$. To minimize $\frac{1}{2}\|w\|^2$, we subject it to

$$y_i(w \cdot x_i) \geq 1.$$

If data is not linearly separable, one possible solution is to penalize each point by distance from margin 1, that is, minimize:

$$\frac{1}{2}\|w\|^2 + \text{constant} \cdot \sum_i \max\{0, 1 - y_i(w \cdot x_i)\} \quad (\mathbf{6}).$$

The result will be complex in low dimensional space, leading to the "curse of dimensionality" as the amount of data needed is often proportional to the number of dimensions n mapped to $O(n^d)$ dimensions [12, 63]. As a result, it will be very expensive in time and memory.

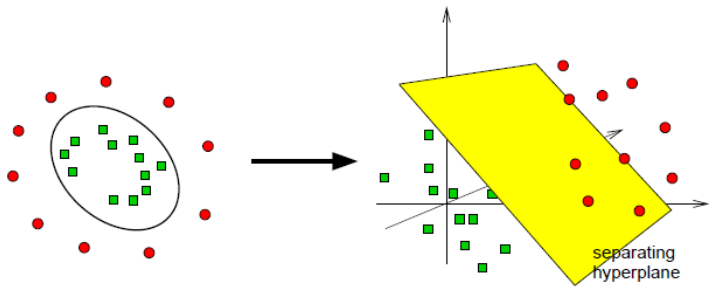


Figure 4. Nonlinear separable data. Left: Complex in low dimensions. Right: Simple in higher dimensions.

A better solution is known as the "kernel trick." The "kernel trick" allows the mapping of data to a higher dimensional space where it becomes linearly separable [62-63]. The construction of the maximal

margin hyperplane in such a space depends on inner products. This solution is very efficient even in large dimensions.

An SVM is a maximal margin hyperplane in the feature space built by using a kernel function in the gene space [62]. For example, given gene expression profiles $p = (p_1, p_{2,\dots,} p_n) \in \mathbb{R}^n$ and $= (q_1, q_{2,\dots,} q_n) \in \mathbb{R}^n$, the similarity in gene space is given by the inner product

$$p \cdot q = p_1 q_1 + p_1 q_1 + \dots + p_n q_n \quad (7).$$

The similarity in the feature space is the kernel function κ (p, q). The four basic kernels are [54]:

Linear: $\kappa\,(p, q) = p^T q$

Polynomial: $\kappa\,(p, q) = (\gamma p^T q + r)^d, \gamma > 0$

Radial Basis Function (RBF): $\kappa\,(p, q) = \exp(-\gamma \|p - q\|^2), \gamma > 0$

Sigmoid: $\kappa\,(p, q) = \tanh(\gamma p^T q + r)$

The parameters of the kernel are γ (the width of RBF coefficient in polynomial (= 1)), d (degree of polynomial), r (additive constant in polynomial (= 0)) and C (error weight or regularization parameter) [54, 63].

Steps in Microarray Data Classification [12]

1. Data normalization.

   Rescale data such that all kernel values fall between -100 and 100, e.g. normalize all entries of the microarray such that they fall between $-10(n)^{1/2}$ and $+10(n)^{1/2}$, where n is the number of expression values per sample.

2. Choosing the regularization parameter C.

   Given the above normalization, the regularization pattern usually does not have much effect. Set it to be $1 < C < 100$.

3. Choosing the kernel.

   For microarray applications, a linear kernel usually suffices as polynomial kernels do not greatly improve performance. The Gaussian kernel is an alternative to the linear kernel if its performance is not so good.

Choosing the SVM Kernel (Similarity Function) [64]

There are many types of SVM kernel functions. Among the commonly used kernels are:

1. No Kernel (Linear Kernel).

   Predict $y = 1$ if $\theta^T x \geq 0$, where $\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n \geq 0$.

   The Linear Kernel uses data transposition. This is the version of the SVM that generates a standard linear classifier. This kernel is used for cases when n, the number of features, is very large, and m, the number of training samples, is very small, or $x \in \mathbb{R}^{n+1}$. This is a reasonable choice of kernel as it avoids fitting (overfitting) a very complicated nonlinear function in a very high dimensional feature space.

2. Gaussian Kernel.

   $f_i = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$, where $l^i = x^i$.

   In this kernel, we need to choose σ^2, considering the bias variance tradeoffs where if σ^2 is large, then we tend to have a higher bias lower variance classifier, and if the σ^2 is small, then we tend to have a lower bias higher variance classifier. The Gaussian Kernel is used when you have the feature space $\mathbb{R}^{\mathrm{n}}$, where n is small and m is large. Feature scaling is needed when using the Gaussian Kernel if the features are on very different scales so that the SVM would give comparable amount of attention to all the features, instead of being dominated only by the feature with the highest scale.

   The kernel or similarity function is used to compute a particular feature of the kernel. For example,

function f = kernel (x1, x2)

f = exp $\left(-\frac{\|x1-x2\|^2}{2\sigma^2}\right)$

return

Where the function takes as input x1, the testing sample $x^{(i)}$, and x2, the training sample $l^{(j)} = x^{(j)}$ to compute f_i. Then, based from this function, it will generate all the features $x \rightarrow \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$ and train the SVM from there.

SVM Multiclass Classification

SVM is originally designed for binary classification of data. But now it is already extended to accommodate classification of K classes, $y \in \{1, 2, 3, \dots, K\}$. Some methods for multiclass classification are as follows:

1. One-vs-All Method.

   In this method, we need to train K SVMs, one to distinguish $y = i$ from the rest, for $i = \{1, 2, 3, \dots, K\}$. This will give us K-parameter vectors $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$. Then, we pick class i with the largest $(\theta^i)^T x$.

2. One-vs-One Method.

   This method builds $k(k-1)/2$ classifiers, with each one learning the separation between two classes, where k is a single class. .

PERFORMANCE EVALUATION

Observed error rates can be used to compare the accuracy of classifiers. We need a way to evaluate the relative performance of the classification procedures.

Cross Validation

In v-fold cross-validation, we first divide the training set into v subsets of equal size. Subsequently one subset is tested using the classifier trained on the remaining $v - 1$ subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data which are correctly classified [54, 65].

Re-Randomization

The cross-validation procedure can prevent the overfitting problem, yet it may not be sufficient, because relatively small cross-validated errors can be achieved by capitalizing on chance properties. A further step to protect against overfitting is to do re-randomize the entire data and then repeat the modeling and validation steps [65]. Re-randomization helps stabilize prediction errors.

IV. DESIGN AND IMPLEMENTATION

DATASET

An oligonucleotide microarray dataset corresponding to 12,600 transcript sequences of a total of 203 samples of lung tumor gene data is used. The data includes 127 adenocarcinomas (ADCA), 21 squamous cell carcinomas (SQ), 20 pulmonary carcinoid (COID), 6 small cell lung carcinomas (SCLC), and 17 normal lung samples. Another 12 adenocarcinomas are suspected to be extrapulmonary metastases.

The dataset is used to generate two smaller datasets, training set and testing set that contains normal and lung cancer data (ADCA, SQ, SCLC, and COID) and ADCA subtypes data.

The dataset has been used in Bhattacharjee et al's [6] and can be found in the PNAS website.

The data inputted to the system must be in comma delimited format, composed of rows containing samples, and columns corresponding to genes. The first row of the training dataset should contain the gene code for each gene and each sample should contain a string label for the classification of the sample. Each testing set should have a gene code for each gene and does not need a column for the class labels. Ideally, the same genes are used for both the training and testing set. The datasets should not contain any missing data.

SYSTEM

The application is implemented as defined in the succeeding diagrams. Generally, the application's input and output requirements are defined by the context diagram below:

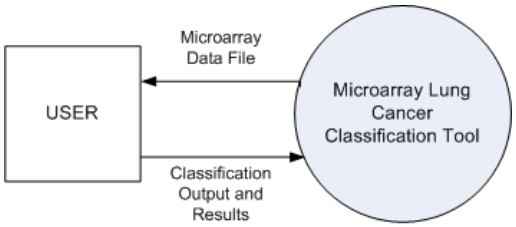


Figure 5. Context Diagram, LCCT

The proposed system takes as input oligonucleotide microarray data corresponding to mRNA expression levels in human lung tumor specimen. The system reads the inputted dataset file and collects the microarray dataset for processing. The user may also input the preprocessing methods and gene selection methods and parameters. If the user does not specify these parameters, the system uses decimal scale normalization for preprocessing and median absolute deviation for gene selection. The system selects by default 1000 genes and 100 marker genes respectively. The user may also submit an optional gene description file to provide more information about the genes in the result. A more detailed illustration of the system's output is shown in the diagram below.

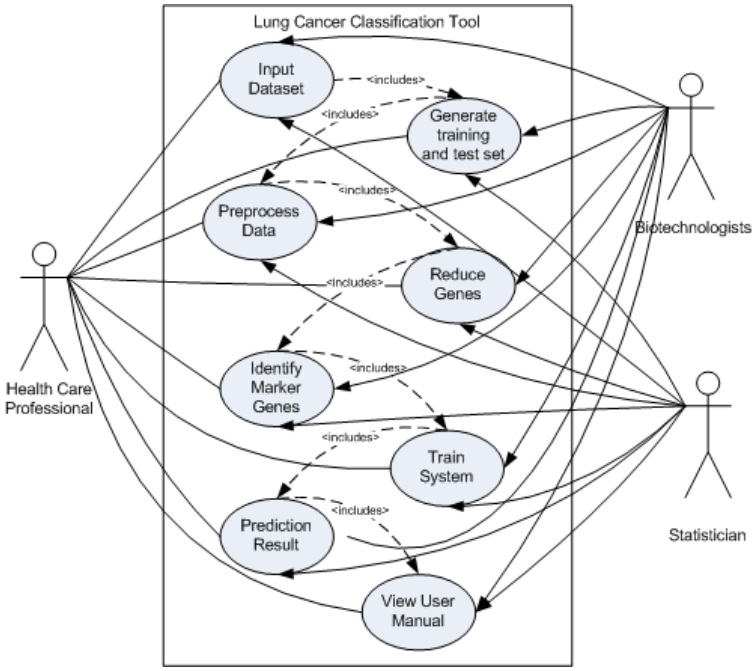


Figure 6. Use Case Diagram, LCCT

The input data is read by the system and training and testing sets are generated using 70% - 30% random division of data. The input data undergoes preprocessing and normalization in order to remove non-biological factors in the data that enables the system to achieve more accurate results. The preprocessed microarray data then undergoes dimension reduction through gene selection in order to remove noise and select the most informative genes relevant to the classification task. Also, the system identifies the marker genes most attributable to the specific lung condition. Finally, the selected features or genes are used in the prediction of the presence or absence of lung cancer in the patient using Support Vector Machine (SVM) classifier. The top level flow of the process is shown below.

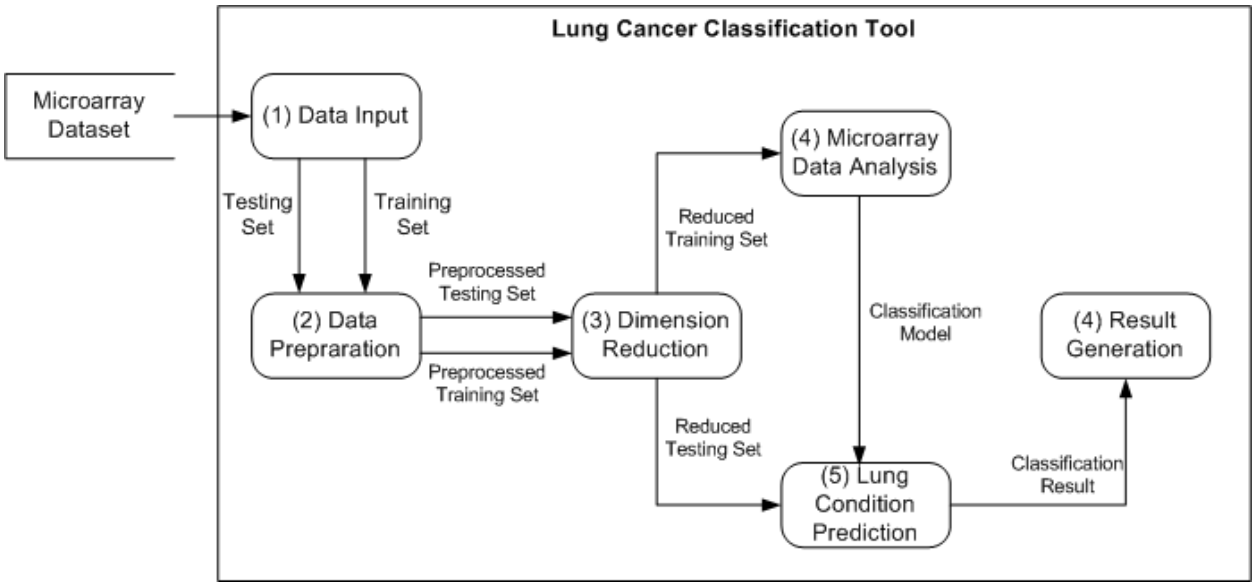


Figure 7. Top Level Data Flow Diagram

The system is composed of five main processes, namely 1) Data Input, 2) Data Preparation, 3) Dimension Reduction, and 4) Microarray Data Analysis, and 5) Lung Condition Prediction, and 6) Result Generation.

If the system has not yet been trained, the User must input a microarray dataset in comma delimited (CSV) format to the system before it can generate training and testing sets to predict the classes of testing data in the Data Input process. The input dataset must include the class names of each sample, along with the names of the subclasses if there are any separated by a hyphen from the parent class. The training sets involve NORMAL, COID, SQ, SCLC and ADCA classes. ADCA subclasses include C1, C2, C3, C4, CM, and GRP1.

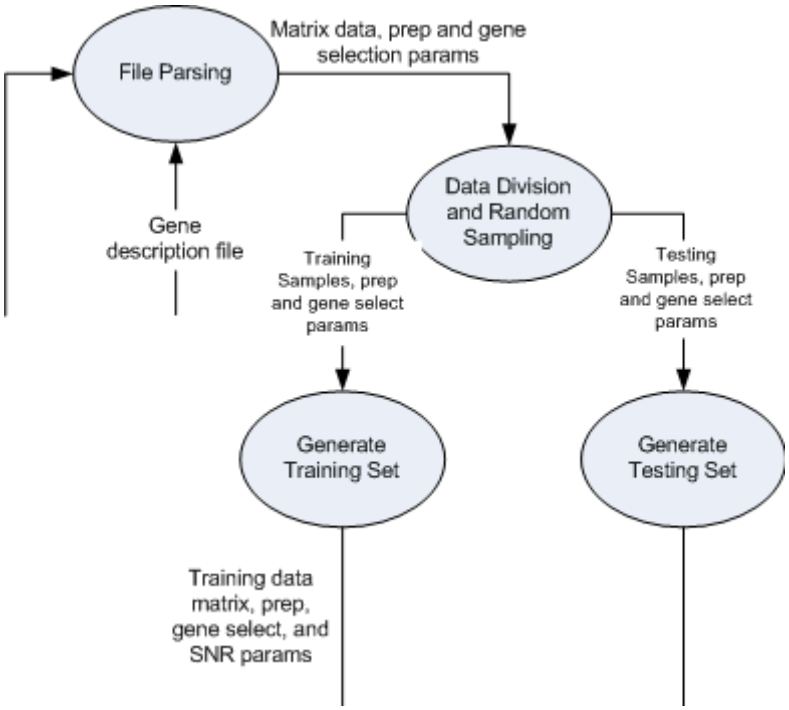


Figure 8. Process (1): Data Input Process, LCCT

The Data Input process gathers information about the dataset that are needed in the succeeding processes. It also allows the User to pick the preprocessing methods that will be applied to the input datasets. The system can perform four preprocessing methods, namely 1) normalization by decimal scaling, 2) quantile normalization, 3) z-score transformation and 4) min-max normalization. The system has a checkbox for these preprocessing methods to aid the selection of the User. If the User does not specify the preprocessing method(s), the system defaults to normalization by decimal scaling.

After the dataset and the parameters are inputted, the system parses the dataset file to collect the data matrix. It then proceeds to divide the data to 70% samples eligible for training and 30% samples eligible for testing. The system picks these samples using random sampling without replacement. From this division, the system proceeds determine the number of training and testing samples are needed and randomly picks from the divided datasets. This process is shown in Figure 8.

The Preprocessing step involves normalizing and transforming data to remove the non-biological factors present. The data matrices generated from the Data Input process undergoes the preprocessing methods specified by the User. This step results to preprocessed data that will be used for dimension reduction, as described below

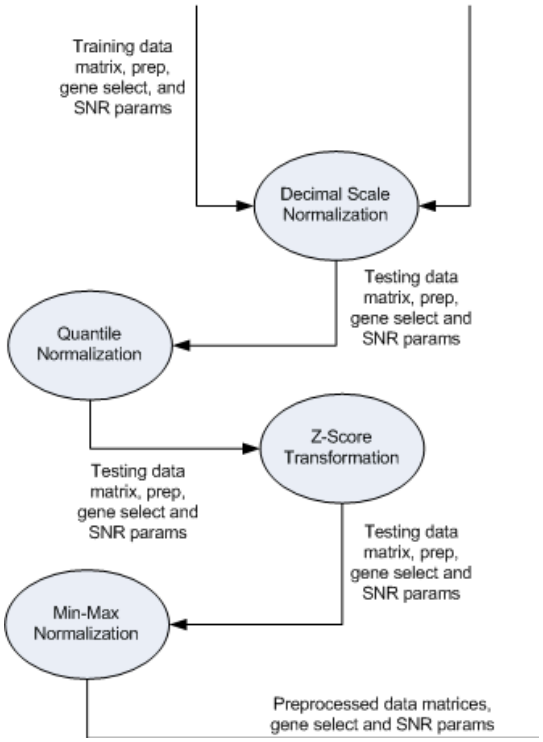


Figure 9. Process (2): Data Preparation

.

The Dimension Reduction step selects the most informative genes in the dataset. The Median Absolute Deviation statistic is used to rank and gather the most differentially expressed ones. The Signal-

to-Noise ratio statistic is used to identify the marker genes that are most attributable to a specific lung condition. The system allows the User to specify the threshold number of genes selected for classification. The Dimension Reduction step output the list of ranked selected genes and the ranked marker genes to the User and the reduced dataset to the next process.

The Microarray Data Analysis process involves training the system and generating classification rules for prediction. The process takes as input the reduced dataset from the previous step. The system builds a classifier model from the generated training data based on the User's specification of whether to classify or sub-classify the inputted dataset. The system builds a model for predicting ADCA, COID, SQ, and SCLC classes if the user opts to only classify the dataset. Otherwise, the system generates a model for predicting the above mentioned classes as well as the ADCA subclasses C1, C2, C3, C4, CM and GRP1.

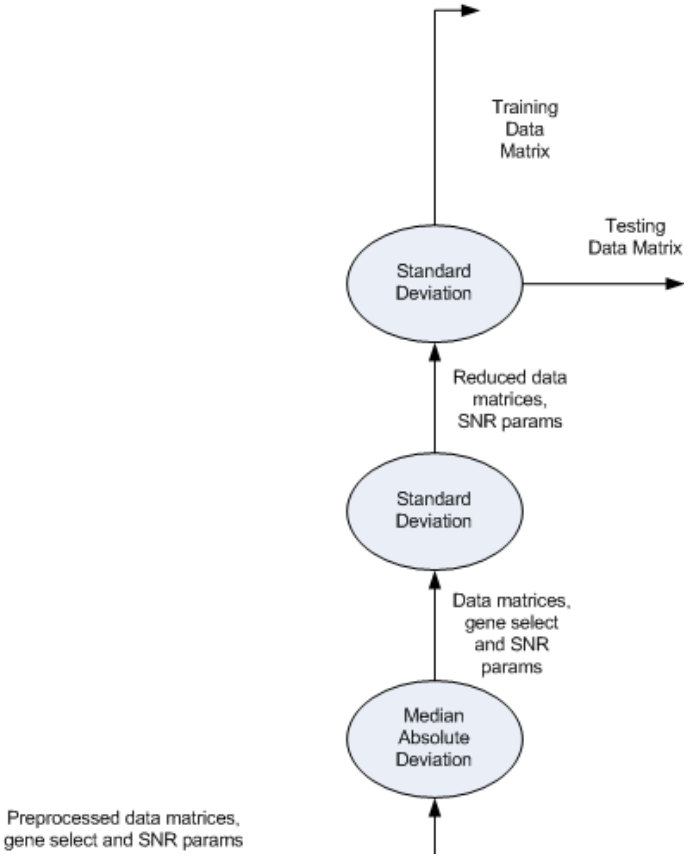


Figure 10. (Process 3): Dimension Reduction Process

Then, the classification rule generated by the Microarray Data Analysis step is used in the Lung Condition Prediction step. If the user opts to sub-classify ADCA into its subclasses, he/ she agree to risk propagating the false prediction rate of the system.

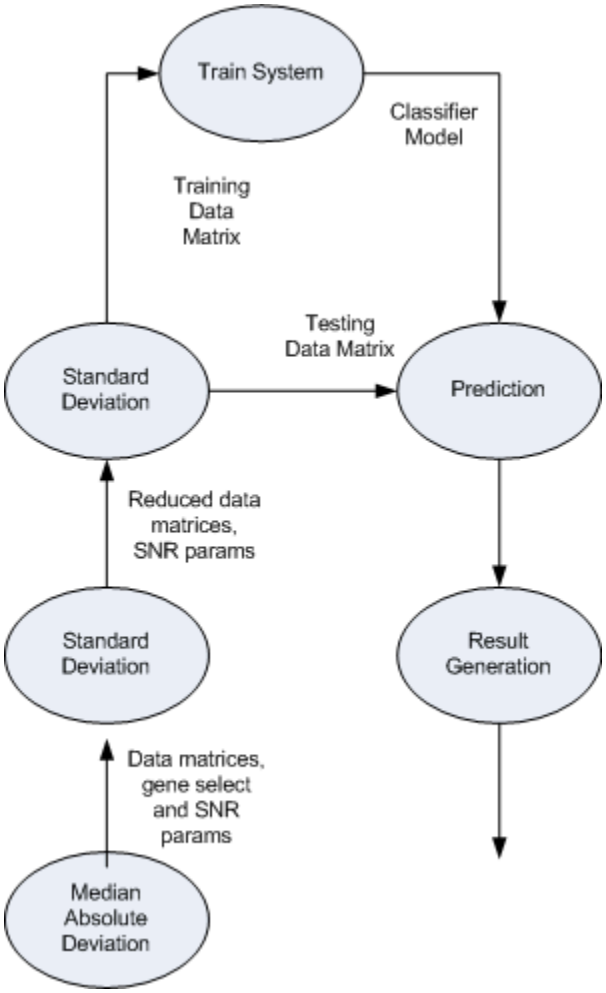


Figure 11. Process (3), Process (4) and Process (5)

Classification result from the training process is used to output the results to the User. The final output of the system is the probable lung condition of the patient based on his/her mRNA expression results. Performance measures and statistics of the process are also outputted to the User. Values computed along the process may be included in the general report of the process.

SVM Implementation

Support Vector Machines is implemented using the LIBSVM library for Java [66]. The chosen regularization parameter C is any integer between 1 and 100, $1 < C < 100$ [6].The kernel or similarity function used is the Linear Kernel (no kernel) as the training datasets used have small sample size and large number of features. LIBSVM also supports multiclass classification using the One-vs-One (OVO) method and it is utilized for the two multiclass classifiers required by the system.

TECHNICAL ARCHITECTURE

The system is built using the following:

- Eclipse Kepler IDE
- Java Runtime Environment (JRE7)
- Java Development Kit (1.6.0_20)
- Windows 7 Operating System
- LibSVM version 3.17

The following minimum system requirements allows for a seamless run of the proposed system:

- Java Runtime Environment (JRE) (constantly updated)
- 1 GB RAM
- 1 GB free hard disk space
- 2.4 MHz single core processor

V. RESULTS

The main window of the application consists of a menu bar, a *Control Panel*, and the *Results Tabs* in the main area. The menu bar contains the *Help* menu where an unfamiliar User can open the *Tutorial and User Guide* window, as shown in Figure 9. The *Control Panel* is on the left side area of the main screen. It consists of the input for datasets and classification controls of the system. Finally, the main area is where the results of classification are displayed. The application main window is shown in Figure 10.

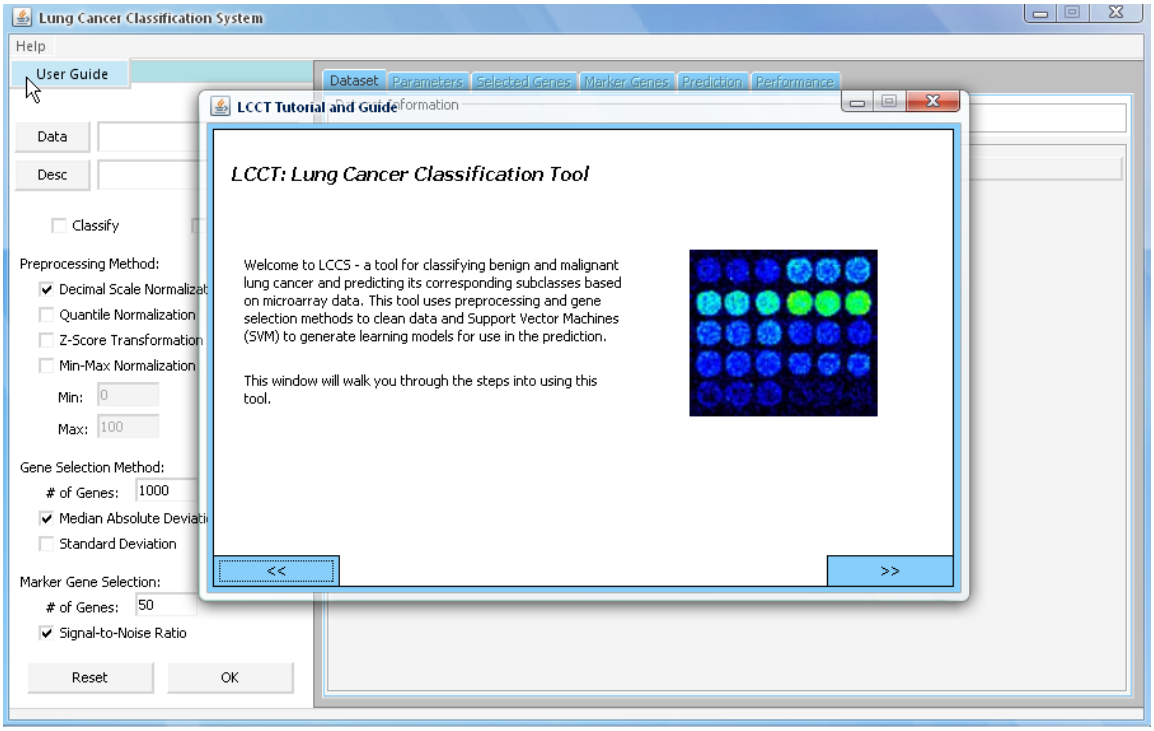


Figure 12. Tutorial and User Guide Window, LCCT

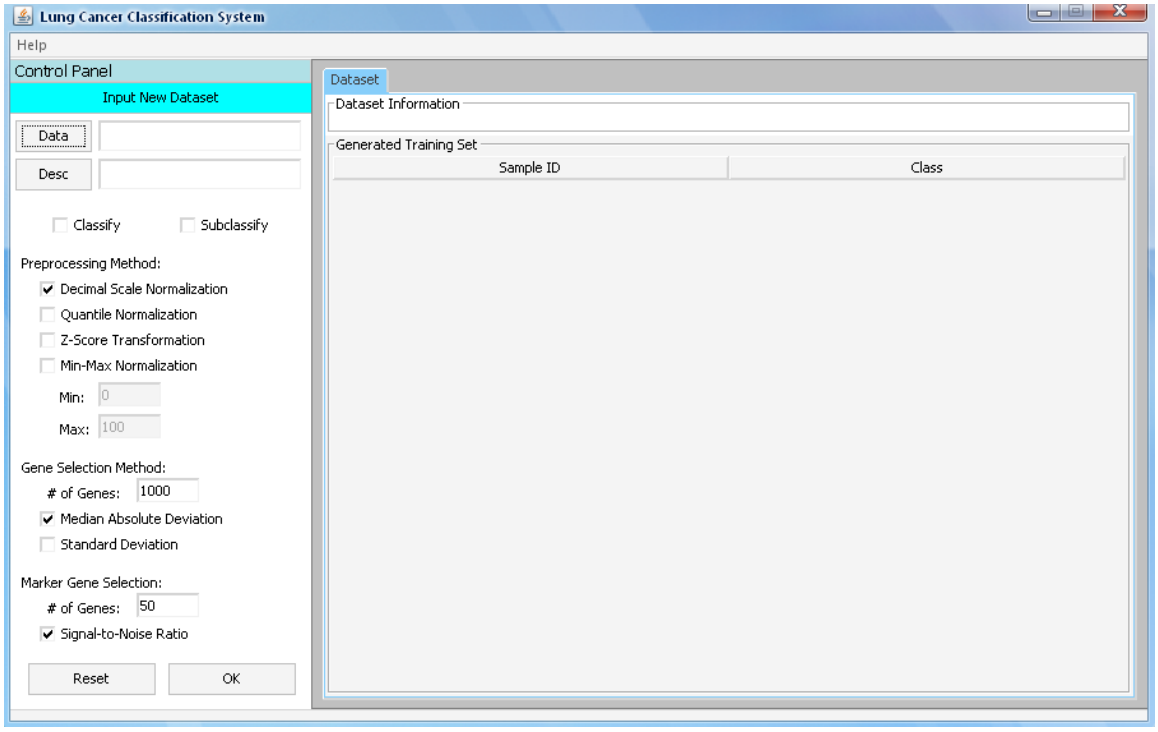


Figure 13. Main Window, LCCT

When the classification process has not yet started, the *Results Tabs* in the main area shows a single active tab, the *Dataset* tab, which immediately updates once the classification process is started.

To start the classification process, the User is required to input a dataset into the system. Clicking the *Data* button on the *Control Panel* launches a *File Chooser* window for the User to browse on the dataset. The User may also upload a gene description file using the *Desc* button. Also, the User is required to choose whether to *Classify* or *Subclassify* the input dataset. Other controls such as the preprocessing methods and gene selection method to be used, as well as whether to identify marker genes may be specified by the User, as shown in Figure 11.

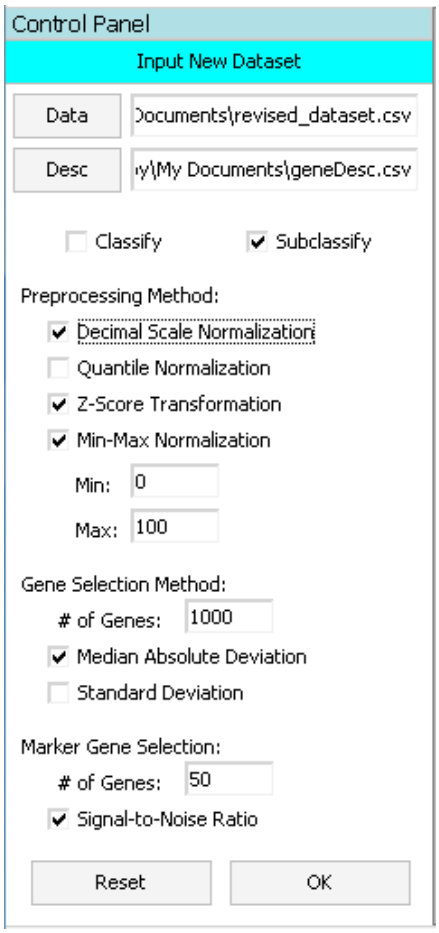


Figure 14. Control Panel, LCCT

Once the input dataset is imported and the controls specified by the User, the *OK* button starts the classification process and results are immediately be displayed on the *Results Tabs* in the main area. The *Dataset* tab, *Parameters* tab, *Selected Genes* tab, *Marker Genes* tab, *Prediction* tab, and *Performance* tab appear one by one as the results are being generated by the system. The result tabs are shown in the following images.

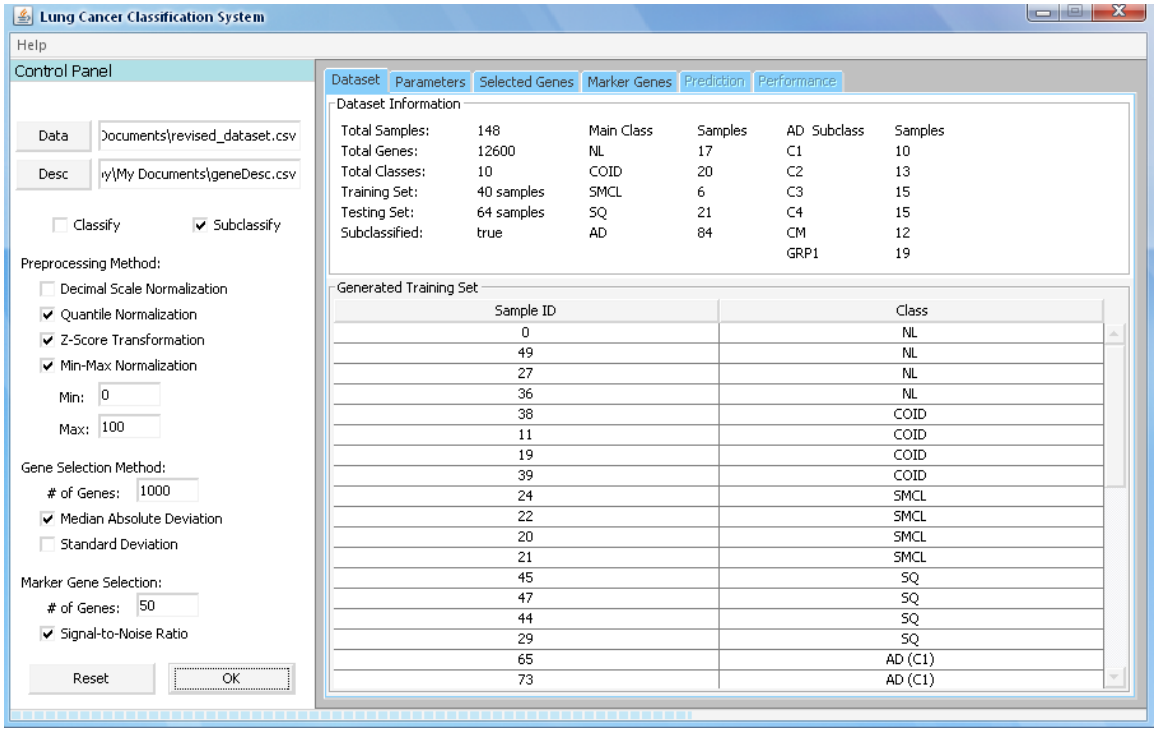
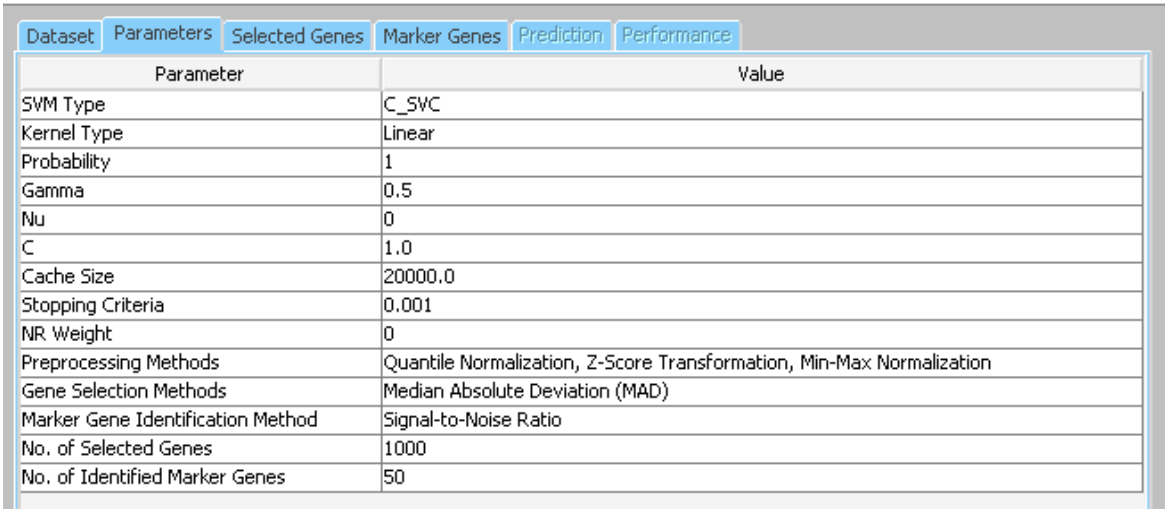


Figure 15. Dataset tab, Results, LCCT

The *Dataset* tab contains information about the imported dataset such as the number of samples, number of genes, number of classes, the class names, and other details. It also shows the number of samples in the generated training set and testing set. The selected training set is also shown.

The *Parameters* tab, on the other hand, shows the SVM parameters used by the system as well as the methods and numbers employed in the classification process, as shown in Figure 13.

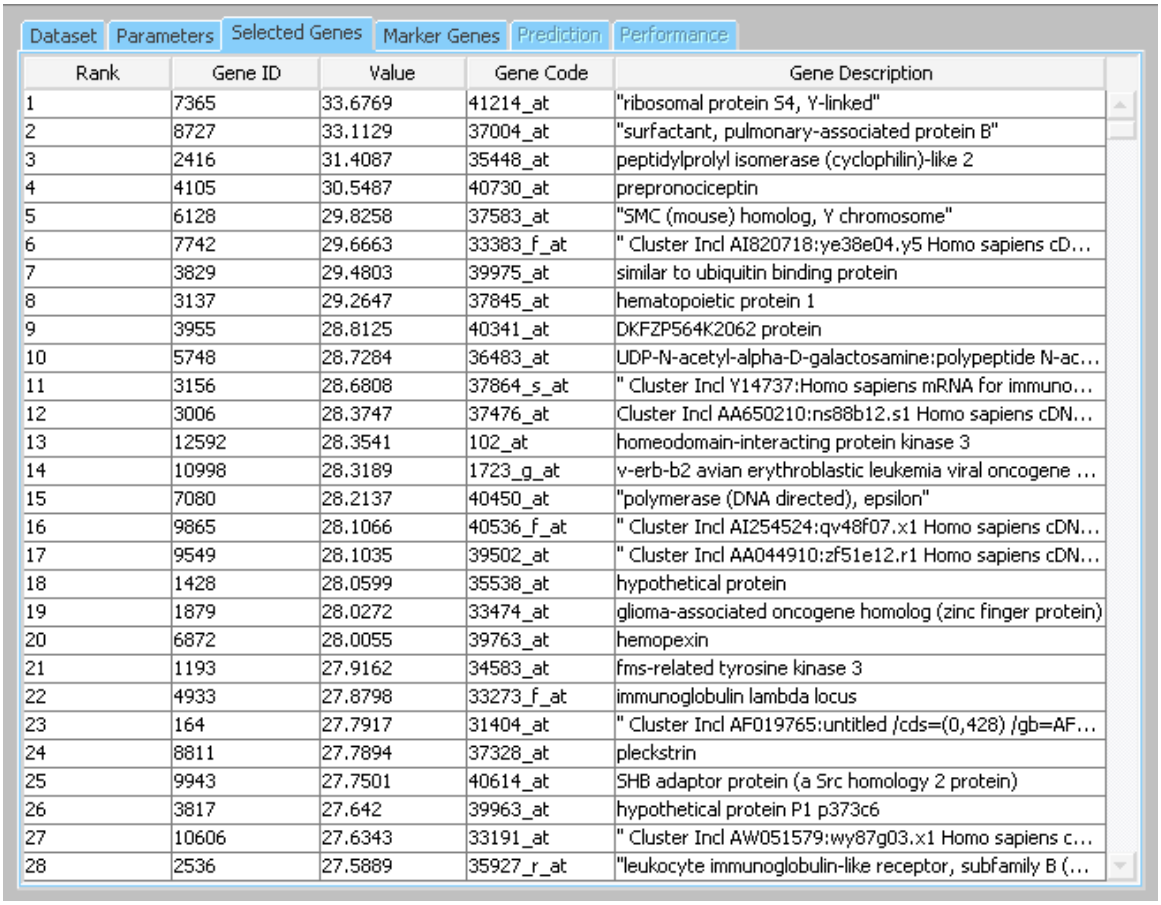
The *Selected Genes* tab, shown in Figure 14, shows the details about the genes selected in the Dimension Reduction stage of the classification process. It shows the number of genes specified by the user, their ranking, statistic, gene code, and description.



Dataset | Parameters | Selected Genes | Marker Genes | Prediction | Performance

Parameter	Value
SVM Type	C_SVC
Kernel Type	Linear
Probability	1
Gamma	0.5
Nu	0
C	1.0
Cache Size	20000.0
Stopping Criteria	0.001
NR Weight	0
Preprocessing Methods	Quantile Normalization, Z-Score Transformation, Min-Max Normalization
Gene Selection Methods	Median Absolute Deviation (MAD)
Marker Gene Identification Method	Signal-to-Noise Ratio
No. of Selected Genes	1000
No. of Identified Marker Genes	50

Figure 16. Parameters Tab, Results, LCCT

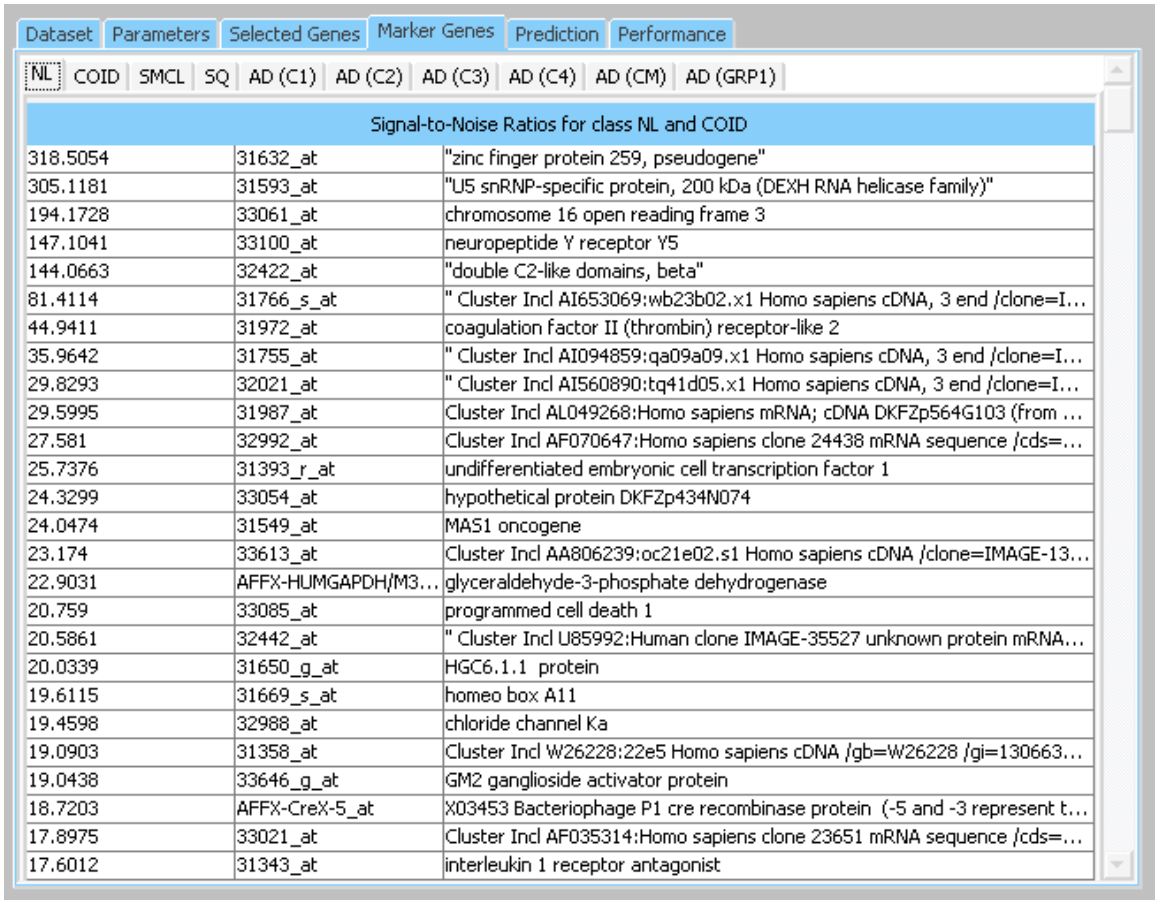


Dataset | Parameters | Selected Genes | Marker Genes | Prediction | Performance

Rank	Gene ID	Value	Gene Code	Gene Description
1	7365	33.6769	41214_at	"ribosomal protein S4, Y-linked"
2	8727	33.1129	37004_at	"surfactant, pulmonary-associated protein B"
3	2416	31.4087	35448_at	peptidylprolyl isomerase (cyclophilin)-like 2
4	4105	30.5487	40730_at	prepronociceptin
5	6128	29.8258	37583_at	"SMC (mouse) homolog, Y chromosome"
6	7742	29.6663	33383_f_at	" Cluster Incl AI820718:ye38e04.y5 Homo sapiens cD...
7	3829	29.4803	39975_at	similar to ubiquitin binding protein
8	3137	29.2647	37845_at	hematopoietic protein 1
9	3955	28.8125	40341_at	DKFZP564K2062 protein
10	5748	28.7284	36483_at	UDP-N-acetyl-alpha-D-galactosamine:polypeptide N-ac...
11	3156	28.6808	37864_s_at	" Cluster Incl Y14737:Homo sapiens mRNA for immuno...
12	3006	28.3747	37476_at	Cluster Incl AA650210:ns88b12.s1 Homo sapiens cDN...
13	12592	28.3541	102_at	homeodomain-interacting protein kinase 3
14	10998	28.3189	1723_g_at	v-erb-b2 avian erythroblastic leukemia viral oncogene ...
15	7080	28.2137	40450_at	"polymerase (DNA directed), epsilon"
16	9865	28.1066	40536_f_at	" Cluster Incl AI254524:qv48f07.x1 Homo sapiens cDN...
17	9549	28.1035	39502_at	" Cluster Incl AA044910:zf51e12.r1 Homo sapiens cDN...
18	1428	28.0599	35538_at	hypothetical protein
19	1879	28.0272	33474_at	glioma-associated oncogene homolog (zinc finger protein)
20	6872	28.0055	39763_at	hemopexin
21	1193	27.9162	34583_at	fms-related tyrosine kinase 3
22	4933	27.8798	33273_f_at	immunoglobulin lambda locus
23	164	27.7917	31404_at	" Cluster Incl AF019765:untitled /cds=(0,428) /gb=AF...
24	8811	27.7894	37328_at	pleckstrin
25	9943	27.7501	40614_at	SHB adaptor protein (a Src homology 2 protein)
26	3817	27.642	39963_at	hypothetical protein P1 p373c6
27	10606	27.6343	33191_at	" Cluster Incl AW051579:wy87g03.x1 Homo sapiens c...
28	2536	27.5889	35927_r_at	"leukocyte immunoglobulin-like receptor, subfamily B (...

Figure 17. Selected Genes Tab, Results, LCCT

The *Marker Genes* tab shown in Figure 15 shows the details about the marker genes identified by the system for each lung condition. It displays the number of genes marker genes to be identified as specified by the user in the *Control Panel*.



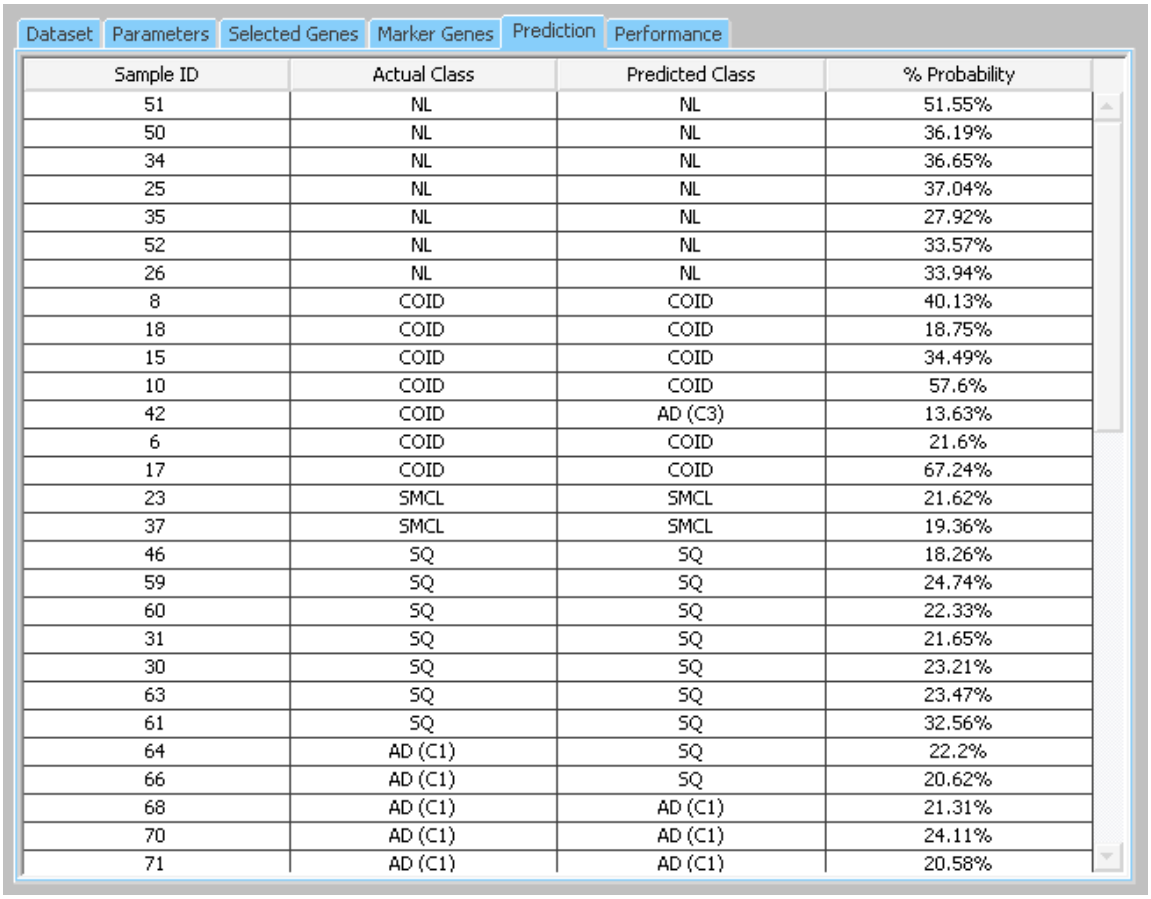
Signal-to-Noise Ratios for class NL and COID

318.5054	31632_at	"zinc finger protein 259, pseudogene"
305.1181	31593_at	"U5 snRNP-specific protein, 200 kDa (DEXH RNA helicase family)"
194.1728	33061_at	chromosome 16 open reading frame 3
147.1041	33100_at	neuropeptide Y receptor Y5
144.0663	32422_at	"double C2-like domains, beta"
81.4114	31766_s_at	" Cluster Incl AI653069:wb23b02.x1 Homo sapiens cDNA, 3 end /clone=I...
44.9411	31972_at	coagulation factor II (thrombin) receptor-like 2
35.9642	31755_at	" Cluster Incl AI094859:qa09a09.x1 Homo sapiens cDNA, 3 end /clone=I...
29.8293	32021_at	" Cluster Incl AI560890:tq41d05.x1 Homo sapiens cDNA, 3 end /clone=I...
29.5995	31987_at	Cluster Incl AL049268:Homo sapiens mRNA; cDNA DKFZp564G103 (from ...
27.581	32992_at	Cluster Incl AF070647:Homo sapiens clone 24438 mRNA sequence /cds=...
25.7376	31393_r_at	undifferentiated embryonic cell transcription factor 1
24.3299	33054_at	hypothetical protein DKFZp434N074
24.0474	31549_at	MAS1 oncogene
23.174	33613_at	Cluster Incl AA806239:oc21e02.s1 Homo sapiens cDNA /clone=IMAGE-13...
22.9031	AFFX-HUMGAPDH/M3...	glyceraldehyde-3-phosphate dehydrogenase
20.759	33085_at	programmed cell death 1
20.5861	32442_at	" Cluster Incl U85992:Human clone IMAGE-35527 unknown protein mRNA...
20.0339	31650_g_at	HGC6.1.1 protein
19.6115	31669_s_at	homeo box A11
19.4598	32988_at	chloride channel Ka
19.0903	31358_at	Cluster Incl W26228:22e5 Homo sapiens cDNA /gb=W26228 /gi=130663...
19.0438	33646_g_at	GM2 ganglioside activator protein
18.7203	AFFX-CreX-5_at	X03453 Bacteriophage P1 cre recombinase protein (-5 and -3 represent t...
17.8975	33021_at	Cluster Incl AF035314:Homo sapiens clone 23651 mRNA sequence /cds=...
17.6012	31343_at	interleukin 1 receptor antagonist

Figure 18. Marker Genes Tab, Results, LCCT

The *Prediction Tab* shows the prediction results of the generated testing set. It shows the sample, its actual class, the class predicted by the system, and the computed probability of the class prediction.

Finally, the *Performance Tab* shows the performance measures of the SVM classifier generated by the system. It shows details about the data such as the *Confusion Matrix*, and measures such as overall accuracy, and precision, recall, specificity, and negative predictive value per class.



Sample ID	Actual Class	Predicted Class	% Probability
51	NL	NL	51.55%
50	NL	NL	36.19%
34	NL	NL	36.65%
25	NL	NL	37.04%
35	NL	NL	27.92%
52	NL	NL	33.57%
26	NL	NL	33.94%
8	COID	COID	40.13%
18	COID	COID	18.75%
15	COID	COID	34.49%
10	COID	COID	57.6%
42	COID	AD (C3)	13.63%
6	COID	COID	21.6%
17	COID	COID	67.24%
23	SMCL	SMCL	21.62%
37	SMCL	SMCL	19.36%
46	SQ	SQ	18.26%
59	SQ	SQ	24.74%
60	SQ	SQ	22.33%
31	SQ	SQ	21.65%
30	SQ	SQ	23.21%
63	SQ	SQ	23.47%
61	SQ	SQ	32.56%
64	AD (C1)	SQ	22.2%
66	AD (C1)	SQ	20.62%
68	AD (C1)	AD (C1)	21.31%
70	AD (C1)	AD (C1)	24.11%
71	AD (C1)	AD (C1)	20.58%

Figure 19. Prediction Tab, Results, LCCT

Confusion Matrix

	NL	COID	SMCL	SQ	AD (C1)	AD (C2)	AD (C3)	AD (C4)	AD (CM)	AD (GRP
NL	7	0	0	0	0	0	0	0	0	0
COID	0	6	0	0	0	0	1	0	0	0
SMCL	0	0	2	0	0	0	0	0	0	0
SQ	0	0	0	7	0	0	0	0	0	0
AD (C1)	0	0	0	2	4	0	0	0	0	0
AD (C2)	0	1	0	0	2	1	3	0	0	0
AD (C3)	0	0	0	0	0	0	6	0	1	0
AD (C4)	0	1	0	0	1	1	1	0	3	0
AD (CM)	0	0	0	0	0	1	0	5	0	1
AD (GRP1	0	0	0	0	0	1	0	1	1	4

Test Outcome / Predicted Class Condition / True Class Correct Prediction Incorrect Prediction

Figure 20. Confusion Matrix, Performance Tab, Results , LCCT

Performance Measures

Accuracy: 57.81%

Class	Precision	Sensitivy (Recall)	Specificity	Negative Predictive V..
NL	1.0	1.0	1.11	0.01
COID	0.75	0.86	1.19	0.01
SMCL	1.0	1.0	1.3	0.01
SQ	0.78	1.0	1.11	0.01
AD (C1)	0.57	0.67	1.32	0.01
AD (C2)	0.25	0.14	9.223372036854776E16	9.223372036854776E1
AD (C3)	0.55	0.86	9.223372036854776E16	9.223372036854776E1
AD (C4)	0.0	0.0	9.223372036854776E16	9.223372036854776E1
AD (CM)	0.0	0.0	9.223372036854776E16	9.223372036854776E1
AD (GRP1)	0.8	0.57	9.223372036854776E16	9.223372036854776E1

Figure 21. Performance Measures, Performance Tab, Results, LCCT

VI. DISCUSSION

The Lung Cancer Classification Tool is a simple tool for predicting benign or malignant lung cancer and its types based on data from oligonucleotide microarrays. The tool allows the user to input dataset files, train the system, and make predictions using the specific preprocessing and gene selection methods applied by the user.

To prepare the data, the system allows the user to choose among the four available preprocessing methods: Decimal Scale Normalization, Quantile Normalization, Z-Score Transfomation, and Min-Max Normalization. To select the genes and reduce the number of features of the microarray, the tool uses the ranked Median Absolute Deviation values for each gene to determine the most relevant genes. The Standard Deviation is also available in the tool. Finally, to determine the marker genes for each class, the system uses the Signal-to-Noise Ratio statistic to identify the marker genes.

The following dataset:

Dataset (Main dataset consisting of 148 samples, 12600 gene expression values, and 5 classes, and 6 Adenocarcinoma subclasses):

- 17 Normal Lung (NL) samples,
- 84 Adenocarcinoma (AD) samples,
- 20 Pulmonary Carcinoid (COID) samples,
- 6 Small Cell Lung Cancer (SMCL) samples,
- 21 Squamous Cell Lung Carcinoma (SQ) samples,
- 10 AD – C1 samples,
- 13 AD – C2 samples,
- 15 AD – C3 samples,
- 15 AD – C4 samples,
- 12 AD – CM (Extrapulmonary Metastases) samples, and
- 19 AD – GRP1 samples.

is inputted to the system and is used by the tool to automatically generate a training set and a testing set using random sampling without replacement. The tool samples the dataset such that each class is represented equally in the resulting training set or testing set to avoid bias to a certain class in the training or prediction process. The system uses a random 70% of the sample as eligible for training and the remaining 30% for testing. Undersampling is used to accommodate the inequality in the number of samples per class.

To validate the accuracy of the prediction of the system, ten-fold cross validation is performed on the classified and subclassified dataset and for each combination of the different preprocessing methods and a threshold of 1000 genes selected through the Median Absolute Deviation (MAD) dimension reduction technique. The following are the results:

Preprocessing Methods	**Accuracy**	
	Classify	**Subclassify**
1	54.774	61.684
2	92.646	89.849
3	85.389	77.462
4	65.218	71.852
1, 2	90.386	87.437
1, 3	82.085	78.817
1, 4	81.932	76.225
2, 3	93.389	90.11
2, 4	87.93	83.02
3, 4	82.011	81.702
1, 2, 3	92.506	89.329
1, 2, 4	91.873	84.774
1, 3, 4	79.537	78.659
2, 3, 4	92.62	90.508
1, 2, 3, 4	89.948	87.221

Legend:

1 Decimal Scale Normalization

2 Quantile Normalization

3 Z-Score Transformation

4 Min-Max Normalization

Figure 22. 10-fold validation results for accuracy using the different preprocessing methods. See Index for full details.

The cross-validation method shows that the output of the system highly depends on the preprocessing method(s) employed and the number of genes to be selected for dimension reduction. Thus, the accuracy of the prediction falls on the choice of the user of which data preprocessing method(s) to use and/or the number of genes for selection.

Based on the results, the Quantile Normalization method yielded the highest average accuracy among the other methods when used in its own, followed by Z-Score Transformation. The Min-Max Normalization and Decimal Scale Normalization yielded the lowest accuracy rates respectively. This is because these two methods involved simple linear rescaling of the data into a more compact range or into a decimal scale respectively. While still preserving the relationship among the original data, they failed to impose the same distribution of gene intensities across samples which the Quantile Normalization and Z-Score Transformation methods did.

While some of the preprocessing methods are not as effective when used in its own, all of the preprocessing method performed efficiently in conjunction with the other preprocessing methods. The lowest accuracy for combined methods is 76.25%, the combination of the two linear preprocessing methods. The best combination of methods is Quantile Normalization and Z-Score Transformation, yielding as high as 93% accuracy.

The number of genes for selection is also a significant factor in the accuracy of prediction. In a five-fold cross-validation test for different numbers of genes for selection using the different preprocessing methods, it is found that the system yields a higher accuracy of prediction when the number of genes selected is larger. The result is also in the discretion of the user on how many genes are to be used for selection. If the number is too small, some important features will be missed out and may lower the accuracy. If the number is too large, the system may consider an unnecessarily large number of features

in the training and prediction, causing the process to take more resources and a significantly lower processing time.

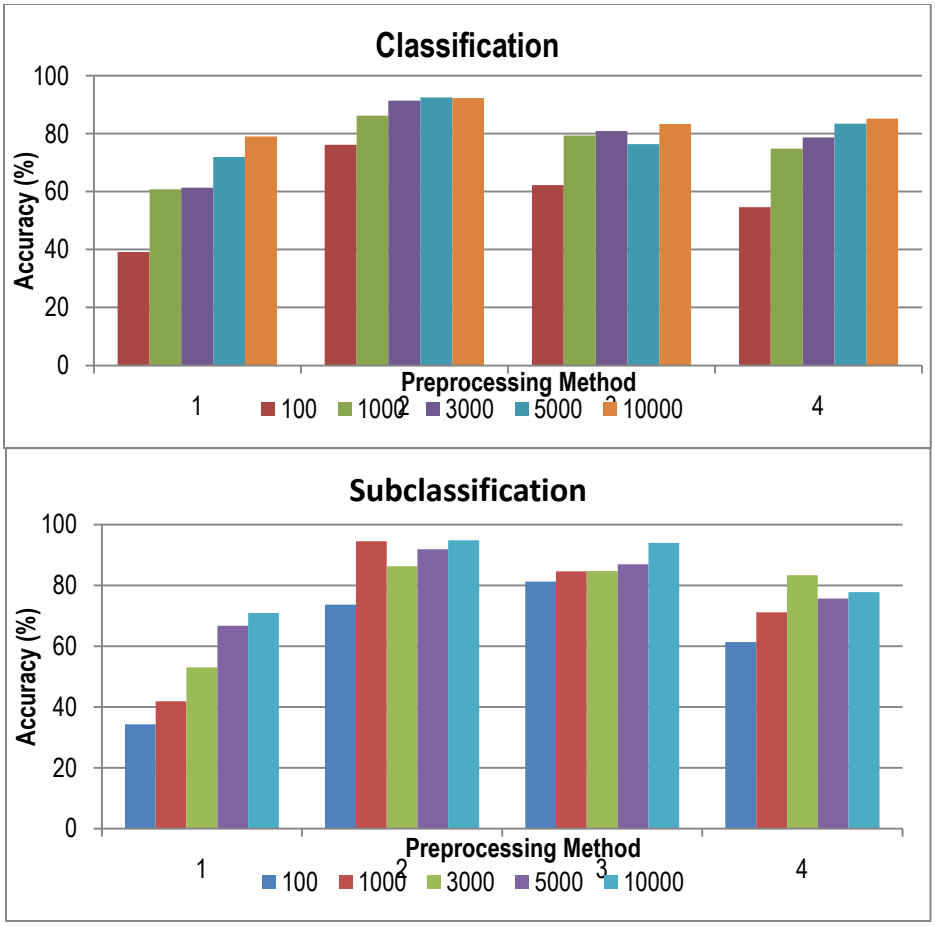
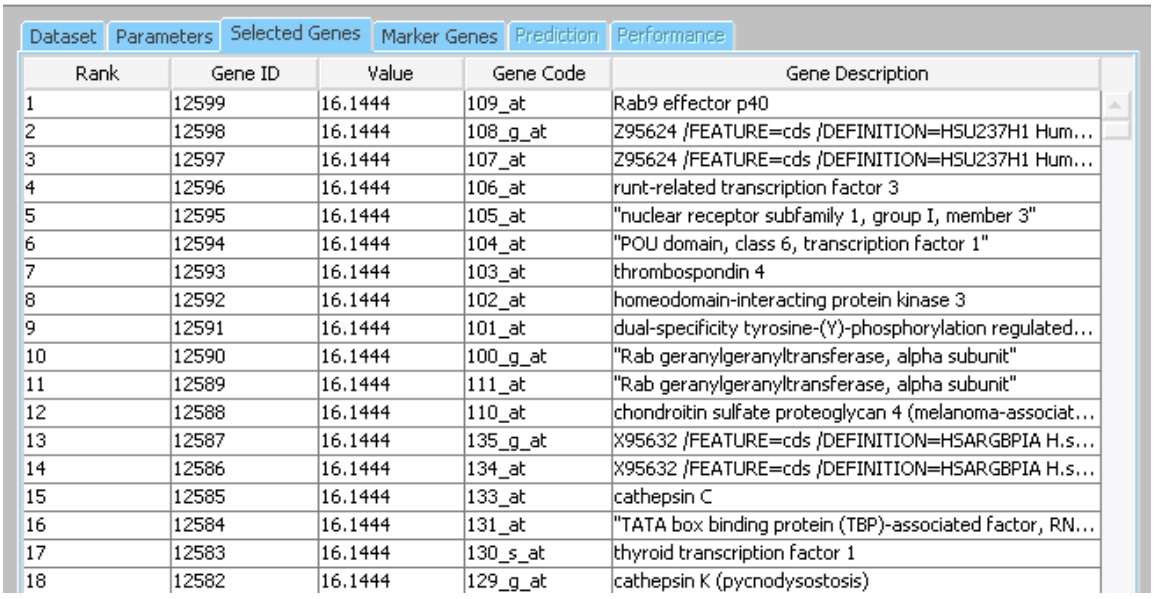


Figure 23. Five-fold cross-validation results for Datasets 1 and 2: Accuracy of prediction against the preprocessing method used and the number of selected genes. See Index for details.

Five-fold cross validation of the results for 100, 1000, 3000, 5000, and 10000 selected genes applied to preprocessed data was generated. Results show that the accuracy is lowest on 100 selected genes, which may be a result of the system missing out other significant genes for prediction. However, the other numbers yield nearly the same accuracies. This means that 1000 to 3000 selected genes are sufficient numbers for prediction. 5000 to 10000 selected genes take unnecessary time and resources to generate a prediction.

Also, it was found that the effect of the Median Absolute Deviation (MAD) and Standard Deviation (SD) ranking method for selecting genes is cancelled out by Quantile Normalization preprocessing, as shown in the Figure below. This is because the Quantile Normalization method imposes the same empirical distribution of expressions across samples and transforms their distribution from one to another, thus effectively making all the genes have the same deviation values. This result shows that the Quantile Normalization method has effectively removed non-biological factors in the data to generate a fairly high accuracy rate, as shown above, even without the effect of Median Absolute Deviation.



Rank	Gene ID	Value	Gene Code	Gene Description
1	12599	16.1444	109_at	Rab9 effector p40
2	12598	16.1444	108_g_at	Z95624 /FEATURE=cds /DEFINITION=H5U237H1 Hum...
3	12597	16.1444	107_at	Z95624 /FEATURE=cds /DEFINITION=H5U237H1 Hum...
4	12596	16.1444	106_at	runt-related transcription factor 3
5	12595	16.1444	105_at	"nuclear receptor subfamily 1, group I, member 3"
6	12594	16.1444	104_at	"POU domain, class 6, transcription factor 1"
7	12593	16.1444	103_at	thrombospondin 4
8	12592	16.1444	102_at	homeodomain-interacting protein kinase 3
9	12591	16.1444	101_at	dual-specificity tyrosine-(Y)-phosphorylation regulated...
10	12590	16.1444	100_g_at	"Rab geranylgeranyltransferase, alpha subunit"
11	12589	16.1444	111_at	"Rab geranylgeranyltransferase, alpha subunit"
12	12588	16.1444	110_at	chondroitin sulfate proteoglycan 4 (melanoma-associat...
13	12587	16.1444	135_g_at	X95632 /FEATURE=cds /DEFINITION=H5ARGBPIA H.s...
14	12586	16.1444	134_at	X95632 /FEATURE=cds /DEFINITION=H5ARGBPIA H.s...
15	12585	16.1444	133_at	cathepsin C
16	12584	16.1444	131_at	"TATA box binding protein (TBP)-associated factor, RN...
17	12583	16.1444	130_s_at	thyroid transcription factor 1
18	12582	16.1444	129_g_at	cathepsin K (pycnodysostosis)

Figure 24. Top genes selected by Median Absolute Deviation (MAD) on data preprocessed using Quantile Normalization

VII. CONCLUSION

The Lung Cancer Classification Tool is a simple tool that relies on Support Vector Machines to classify lung cancer data based on oligonucleotide microarrays. It provides preprocessing methods to remove possible non-biological factors and biases in the data. It also provides gene selection techniques to select the most relevant genes and reduced the extremely large size of the microarray dataset. Also, it computes which genes are markers or most attributable to a certain class of lung cancer. Finally, it presents its output and results in a tabular form and provides as much information about the data as possible.

This software can be used to achieve more accurate predictions of lung cancer conditions using learning models generated from training data that are preprocessed and reduced. Also, the output of this system can be used in the analysis and identification of genes that are most attributable to a certain class of the disease. However, given the large dimensions of microarray data, this tool requires a large amount of memory and processor resources for optimum performance. Also, given the limitations of the methods available in the system, improvements are necessary for more accurate results and more extensive analyses required.

VIII. RECOMMENDATIONS

While the tool seems to be effective in classifying types of lung cancer using oligonucelotide microarray data, there are still many features that needs to be improved. Among these is the addition of preprocessing methods suitable for oligonucleotide microarrays. While the system already offers four methods, there are still many methods, new and improved, that can be used to more effectively prepare the data.

Another is the addition of gene selection methods to reduce the large dimensions of the dataset. The methods available in the tool are feature filter methods that score individual features. While filter methods are faster, the disadvantage of this is the identification of the threshold number of features for the data. Wrapper methods, another type of feature selection methods, provide better accuracy results and can be added to the system to provide better results.

For marker gene identification, there are many modifications of the Signal-to-Noise ratio (SNR) statistic that can be extended into the system to be able to provide a much clearer list of marker genes for each lung condition. The SNR statistic used in this tool only involves binary identification and comparison of genes between samples, hence the more scattered results.

IX. BIBLIOGRAPHY

[1] Medical News Today, "What is Lung Cancer?" www.medicalnewstoday.com/info/lung-cancer, *Medical News Today*, n.d. Accessed: April 2013.

[2] MedicineNet.com, "What is lung cancer?" www.medicinenet.com/lung_cancer/article.htm#what_is_lung_cancer, *MedicineNet.*com, 1996-2013. Accessed: April 2013.

[3] PhilamLife, "Lung Cancer: Get the Facts." www.philamlife.com/en/resources/e8204a8049583f81beaabf4e12e54bc2/AIA_embrace_time_brochure_ph.pdf, n.d. Accessed: April 2013.

[4] M. Yu, "PSMO observes World Lung Cancer Awareness month." www.medobserver.com/specialtyarticle.php?ArticleID=341, MedicalObserver, 2011. Accessed: April 2013.

[5] R. Sindico, "What makes Asian women more prone to lung cancer?" www. philstar.com/health-and-family/2012/11/14/866659/what-makes-asian-women-more-prone-lung-cancer, Philstar.com, 2012. Accessed: April 2013.

[6] A. Bhattacharjee, W.G. Richards, J. Staunton, C. Li, S. Monti, P. Vasa, C. Ladd, J. Beheshti, R. Bueno, M. Gillette, M. Loda, G. Weber, E.J. Mark, E.S. Lander, W. Wong, B.E. Johnson, T.R. Golub, D.J. Sugarbaker, and M. Meyerson, "Classification of human lung carcinomas by mRNA expression profiling reveals distinct adenocarcinoma subclasses," PNAS, 2001.

[7] M.E. Garber, O.G. Troyanskaya, K. Schluens, S. Petersen, Z. Thaesler, M. Pacyna-Gengelbach, M. van de Rijn, G.D. Rosen, C.M. Perou, R.I. Whyte, R.B. Altman, P.O. Brown, D. Botstein, and I. Petersen, "Diversity of gene expression in adenocarcinoma of the lung," PNAS, 2001.

[8] R. G. Ramani and S.G. Jacob, "Improved Classification of Lung Cancer Tumors Based on Structural and Physicochemical Properties of Proteins Using Data Mining Models," PLoS ONE, 2012.

[9] I-J. Kim, H.C. Kang, S-G. Jang, K. Kim, S-A. Ahn, H-J. Yoon, S-N. Yoon, and J-G. Park, "Oligonucleotide microarray analysis of distinct gene expression patterns in colorectal cancer tissues harboring BRAF and K-ras mutations," Carcinogenesis, 2005.

[10] A. Oberthuer, F. Berthold, P. Warnat, B. Hero, Y. Kahlert, R. Spitz, K. Ernestus, R. Konig, S. Haas, R. Eils, M. Schwab, B. Brors, F. Westermann, and M. Fischer, "Customized Oligonucleotide Microarray Gene Expression-Based Classification of Neuroblastoma Patients Outperforms Current Clinical Risk Stratification," Journal of Clinical Oncology, 2006.

[11] W.S. Noble, "What is a support vector machine?" Nature Publishing Group, pp. 1565-1567, 2006. Accessed: May 2013.

[12] S. Mukherjee, "Classifying Microarray Data Using Support Vector Machines" n.p., n.d.

[13] A.V. Gomes, L.J.K. Wee, A.M. Khan, L.H.V.G. Gil, E.T.A. Marques, Jr, C.E. Calzavara-Silva, T.W. Tan, “Classification of Dengue Fever Patients Based on Gene Expression Data Using Support Vector Machines,” PLoS ONE, 2010.

[14] M. Sanchez-Carbayo, N.D. Socci, J. Lozano, F. Saint, and C. Cordon-Cardo, “Defining Molecular Profiles of Poor Outcome in Patients with Invasive Bladder Cancer Using Oligonucleotide Microarrays,” *Journal of Clinical Oncology*, 2006.

[15] R. Sandhu, MBBS, J.S. Parker, MS, W.D. Jones, PhD, C.A. Livasy, MD, and W.B. Coleman, PhD, “Microarray-Based Gene Expression Profiling for Molecular Classification of Breast Cancer and Identification of New Targets for Therapy,” *LabMedicine*, June 2010.

[16] I. LG. Newton and G. Rueselers, “The effect of training set on the classification of honey bee guy microbiota using Naïve Bayesian Classifier,” *BMC Microbiology*, 2012.

[17] P. Chopra, J. Lee, J. Kang, and S. Lee, “Improving Cancer Classification Accuracy Using Gene Pairs,” *PLoS ONE*, December 2010.

[18] O. Dagliyan, F. Uney-Yuksektepe, I. H. Kavakli, and M. Turkay, “Optimization Based Tumor Classification from Microarray Gene Expression Data,” *PLoS ONE*, February 2011.

[19] I. Fishel, A. Kaufman, and E. Ruppin, “Meta-analysis of gene expression data: a predictor-based approach,” *Bioinformatics*, April 2007.

[20] B. Li, C-H. Zheng, D-S. Huang, L. Zhang, and K. Han, “Gene expression data classification using locally linear discriminant embedding,” *Elsevier*, 2010.

[21] X. Wang, and R. Simon, “Microarray-based cancer prediction using single genes,” *BMC Bioinformatics*, 2011.

[22] G. Zararsiz, F. Elmali, A. Ozturk, “Bagging Support Vector Machines for Leukemia Classification,” *International Journal of Computer Science Issues*, November 2012.

[23] J-H Hong, J-K Min, U-K, Cho and S-B, Cho, “Fingerprint classification using one-vs-all support vector machines dynamically ordered with naïve Bayes classifiers,” *Elsevier*, 2008.

[24] D. Anguita, S. Ridella, and D. Sterpi, “Testing the Augmented Binary Multiclass SVM on Microarray Data,” *IEEE*, n.d.

[25] H.L. Yu, S. Gao, B. Qin, and J. Zhao, “Multiclass microarray data classification based on confidence evaluation,” *Genetics and Molecular Research*, May 2012.

[26] G. Gomez, “Normalization methods and data processing,” *Bioinformatics Unit CNIO*, May 2010.

[27] “Data Mining: Concepts and Techniques,” pp. 30-31, n.p, n.d.

[28] B.M. Bolstad, R.A. Irizarry, M. Astrand, and T.P. Speed, “A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Variance and Bias,” *Bioconductor*, 2002.

[29] H. Hu, J. Li, H. Wang, and G, Daggard, “Combined Gene Selection Methods for Microarray Data Analysis,” n.p., 2004.

[30] C. Cheadle, M.P. Vawter, W.J. Freed, and K.G. Becker, “Analysis of Microarray Data Using Z Score Transformation,” *Journal of Medical Diagnostics*, May 2003.

[31] D. Mishra and B. Sahu, “Feature Selection for Cancer Classification: Signal-to-noise Ratio Approach,” *International Journal of Scientific & Engineering Research*, April 2011.

[32] D. Venet, V. Detours, and H. Bersini, “A Measure of Signal-to-Noise Ratio of Microarray Samples and Studies Using Gene Correlations,” *PLoS ONE*, December 2012.

[33] K.J. Savage, S. Monti, J.L. Kutok, G. Cattoretti, D. Neuberg, L. de Laval, P. Kurtin, P. Dal Cin, C. Ladd, F. Feuerhake, R. Aguiar, S. Li, G. Salles, F. Berger, W. Jing, G. Pinkus, T. Habermann, R. Dalla-Favera, N. Harris, J.C. Aster, T. Golub, and M. Shipp, “Mediastinal Large B-Cell Lymphoma: A unique subset of diffuse large B-cell lymphoma with an expression signature resembling that of Hodgkin’s Reed-Sternberg Cells,” n.p., n.d.

[34] R. Hoffman, T. Seidl, and M. Dugas, “Profound effect of normalization on detection of differentially expressed genes in oligonucleotide microarray data analysis,” *Genome Biology*, June 2002.

[35] About.com, “Lung Adenocarcinoma,” lungcancer.about.com/od/typesoflungcancer/a/Lung-Adenocarcinoma.htm, *About.com*, 2013. Accessed: May 2013.

[36] www.cap.org/apps/docs/reference/myBiopsy/LungSquamousCellCarcinoma.pdf. Accessed: April 2013.

[37] “Pulmonary Carcinoid,” endocrinediseases.org/neuroendocrine/pulmonary_carcinoid.shtml, *The American Association of Endocrine Surgeons Patient Education Site*, n.d. Accessed: May 2013.

[38] A. Mandal, “Genes – What are Genes?” www.news-medical.net/health/Genes-What-are-genes.aspx, *News Medical*, n.d. Accessed May 2013.

[39] Genetics Home Reference, “What is a gene?” ghr.nlm.nih.gov/handbook/basics/gene, *US National Library of Medicine*, May 2013. Accessed: May 2013.

[40] Medical News Today, “What is a Gene? What are Genes?” www.medicalnewstoday.com/articles/120574.php, *MediLexicon International Ltd.*, May 2013. Accessed: May 2013.

[41] “Microarrays: Chipping Away at the Mysteries of Science and Medicine,” www.ncbi.nlm.nih.gov/About/primer/microarrays.html, *National Center for Biotechnology Information*, 27 July 2007. Accessed: May 2013.

[42] A. Mandal, “What is DNA?” www.news-medical.net/health/What-is-DNA.aspx, *News Medical*,n.d. Accessed: May 2013..

[43] “Biomarker – What is a Biomarker?” www.news-medical.net/health/Biomarker-What-is-a-Biomarker.aspx, *News Medical*, n.d. Accessed: May 2013.

[44] BME215-10-DNA microarray.pdf. Accessed: May 2013.

[45] Howard Hughes Medical Institute, “How to Analyze DNA Microarray Data,” www.hhmi.org/biointeractive/genomics/microarray_analyzing/01.html, *Biointeractive.org*, n.d. Accessed: 08 May 2013.

[45] S. Dudoit, R. Gentleman, R. Irizarry, and Y. Yang, “Introduction to DNA microarray technologies,” n.p, n.d. Accessed: May 2013.

[46] M. Rouse, “machine learning,” Whatis.techtarget.com/definition/machine-learning, *WhatIs.com*, 199-2013. Accessed: May 2013.

[47] T. Hofmann, “Introduction to Machine Learning,” *Thomas Hofmann*, 2002-2003. Accessed: May 2013.

[48] E. Alpaydin, “Introduction to Machine Learning,” *Massachussetts Institute of Technology,* 2004. Accessed: May 2013.

[49] M. Kim, “Statistical Classification,” n.p, April 2010. Accessed: May 2013.

[50] S. Reddy, “Machine Learning: Classification,” www.grok.in/notes/machine-learning-classification, *The art of Information Engineering*, n.d. Accessed: May 2013.

[51] M. Mandel, “Lecture 3: Machine learning, classification, and generative models,” n.p., February 2008. Accessed: May 2013.

[52] S.B. Kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” *Informatica*, pp. 249-268, 2009. Accessed: May 2013.

[53] R. Schapire, “Machine Learning Algorithms for Classification,” n.p, n.d. Accessed: May 2013.

[54] C. Hsu, C. Chang, and C. Lin, “A Practical Guide to Support Vector Classification,” n.p, April 2010. Accessed: May 2013.

[55] J. Han, M. Kamber, and M. Pei, “Data Mining: Concepts and Techniques,” *Elsevier*, 2011. Accessed: January 2014.

[56] S. Mukherjee, “Classifying Microarray Data Using Support Vector Machines,” n.p, 2002.

[57] L. Ladha and T. Deepa, “Feature Selection Methods and Algorithms,” *Internation Journal on Computer Science and Engineering (IJCSE)*, May 2011.

[58] Y. Saeys, I. Inza, and P. Larrañaga, “A review of feature selection techniques in bioinformatics,” *BioInformatics*, 2007.

[59] D. Chen, Z. Liu, X. Ma, and D. Hua, “Selecting Genes by Test Statistics,” *Journal of Biomedicine and BioTechnology*, 2005.

[60] W.S. Noble, “What is support vector machine?” *Nature Publishing Group*, pp. 1565-1567, December 2006. Accessed: May 2013.

[61] D. Srivastava and L. Bhambhu, “Data Classification Using Support Vector Machine,” *Journal of Theoretical and Applied Information Technology*, 2005-2009. Accessed: May 2013.

[62] F. Markowetz, “Classification by Support Vector Machines,” *Practical DNA Microarray Analysis*, 2003. Accessed: May 2013.

[63] R. Schapire, “Machine Learning Algorithms for Classfication,” n.p., n.d. Accessed: May 2013.

[64] “12.6 Machine Learning Using SVM” [Video File], www.youtube.com/watch?v=dzuuNPmGAVU. Accessed: September 2013.

[65] J.J. Dai, L. Lieu, and D. Rocke, “Dimension Reduction for Classificatin with Gene Expression Microarray Data,” *Statistical Applications in Genetics and Molecular Biology*, 2006.

[66] C-C Chang and C-J Lin, “LIBSVM: a library for support vector machines,” http://www.csie.ntu.edu.tw/~cjlin/libsvm, *ACM Transactions on Intelligent Systems and Technology*, pp. 2:27:1—27:27, 2011.

X. APPENDIX

I. Results

A. Cross-Validation Results for Preprocessing Methods

		Dataset 1			Dataset 2				Dataset 1			Dataset 2				Dataset 1			Dataset 2		
	Prep	Train	Test	Acc	Train	Test	Acc	Prep	Train	Test	Acc	Train	Test	Acc	Prep	Train	Test	Acc	Train	Test	Acc
1	1	20	30	66.67	42	38	71.05	2	20	22	90.91	42	39	97.44	3	20	22	81.82	42	39	82.05
2	1	20	34	67.65	42	39	66.67	2	20	38	86.84	42	32	87.5	3	20	34	100	42	35	80
3	1	20	38	34.21	42	38	76.32	2	20	38	84.21	42	28	89.29	3	20	34	88.24	42	38	81.58
4	1	20	38	52.63	42	38	57.89	2	20	26	100	42	39	89.74	3	20	30	73.33	42	32	68.75
5	1	20	26	53.85	42	28	60.71	2	20	22	90.91	42	28	82.14	3	20	34	73.53	42	38	76.32
6	1	20	22	54.55	42	35	68.57	2	20	38	89.47	42	39	82.05	3	20	26	80.77	42	35	77.14
7	1	20	26	50	42	32	62.5	2	20	34	97.06	42	28	85.71	3	20	38	92.11	42	39	79.49
8	1	20	38	50	42	39	51.28	2	20	26	100	42	39	92.31	3	20	30	90	42	28	75
9	1	20	22	50	42	32	53.13	2	20	22	90.91	42	39	97.44	3	20	38	89.47	42	35	82.86
10	1	20	22	68.18	42	39	48.72	2	20	26	96.15	42	39	94.87	3	20	26	84.62	42	28	71.43
Mean				**54.774**			**61.684**				**92.646**			**89.849**				**85.389**			**77.462**
		Dataset 1			**Dataset 2**				**Dataset 1**			**Dataset 2**				**Dataset 1**			**Dataset 2**		
	Prep	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**	**Prep**	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**	**Prep**	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**
1	4	20	26	84.62	42	38	81.58	1,2	20	34	94.12	42	38	100	1,3	20	22	45.45	42	39	79.49
2	4	20	30	73.33	42	32	78.13	1,2	20	22	100	42	28	85.71	1,3	20	30	96.67	42	39	82.05
3	4	20	30	73.33	42	35	77.14	1,2	20	22	95.45	42	32	96.88	1,3	20	22	90.91	42	32	84.38
4	4	20	26	53.85	42	38	68.42	1,2	20	38	89.47	42	28	85.71	1,3	20	38	89.47	42	35	71.43
5	4	20	30	66.67	42	28	78.57	1,2	20	34	97.06	42	28	67.86	1,3	20	30	66.67	42	35	71.43
6	4	20	34	61.76	42	38	86.84	1,2	20	38	94.74	42	35	91.43	1,3	20	22	95.45	42	28	75
7	4	20	30	60	42	35	54.29	1,2	20	34	82.35	42	30	93.33	1,3	20	34	94.12	42	35	82.86

8	4	20	34	35.29	42	32	62.5	1,2	20	38	78.95	42	38	86.84	1,3	20	26	57.69	42	28	85.71
9	4	20	30	73.33	42	35	60	1,2	20	26	92.31	42	38	89.47	1,3	20	26	92.31	42	38	73.68
10	4	20	30	70	42	38	71.05	1,2	20	34	79.41	42	35	77.14	1,3	20	38	92.11	42	28	82.14
Mean				**65.218**			**71.852**				**90.386**			**87.437**				**82.085**			**78.817**
		Dataset 1			**Dataset 2**				**Dataset 1**			**Dataset 2**				**Dataset 1**			**Dataset 2**		
	Prep	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**	**Prep**	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**	**Prep**	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**
1	1,4	20	34	85.29	42	28	42.86	2,3	20	26	96.15	42	28	85.71	2,4	20	30	76.67	42	35	82.86
2	1,4	20	22	86.36	42	35	82.86	2,3	20	34	88.24	42	28	85.71	2,4	20	22	95.45	42	39	94.87
3	1,4	20	34	91.18	42	35	88.57	2,3	20	22	95.45	42	38	92.11	2,4	20	30	76.67	42	28	96.43
4	1,4	20	26	88.46	42	38	68.42	2,3	20	22	95.45	42	35	97.14	2,4	20	34	94.12	42	39	84.62
5	1,4	20	26	65.38	42	35	91.43	2,3	20	22	95.45	42	39	94.87	2,4	20	34	94.12	42	39	79.49
6	1,4	20	34	88.24	42	32	93.75	2,3	20	30	90	42	38	92.11	2,4	20	30	90	42	32	81.25
7	1,4	20	22	68.18	42	28	71.43	2,3	20	30	96.67	42	38	84.21	2,4	20	22	100	42	32	90.63
8	1,4	20	26	92.31	42	39	66.67	2,3	20	34	88.24	42	39	97.44	2,4	20	34	64.71	42	39	53.85
9	1,4	20	30	83.33	42	32	90.63	2,3	20	34	94.12	42	39	87.18	2,4	20	22	95.45	42	39	84.62
10	1,4	20	34	70.59	42	32	65.63	2,3	20	34	94.12	42	39	84.62	2,4	20	38	92.11	42	38	81.58
Mean				**81.932**			**76.225**				**93.389**			**90.11**				**87.93**			**83.02**
		Dataset 1			**Dataset 2**				**Dataset 1**			**Dataset 2**				**Dataset 1**			**Dataset 2**		
	Prep	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**	**Prep**	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**	**Prep**	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**
1	3,4	20	38	97.47	42	39	97.44	1,2,3	20	26	96.15	42	35	97.14	1,2,4	20	30	93.33	42	32	81.25
2	3,4	20	22	90.91	42	28	82.14	1,2,3	20	38	92.11	42	39	94.87	1,2,4	20	30	86.67	42	39	82.05
3	3,4	20	38	89.47	42	32	84.38	1,2,3	20	30	93.33	42	35	94.29	1,2,4	20	26	100	42	35	77.14
4	3,4	20	22	86.36	42	38	84.21	1,2,3	20	26	92.31	42	39	82.05	1,2,4	20	26	96.15	42	35	94.29
5	3,4	20	34	79.41	42	35	68.57	1,2,3	20	22	100	42	32	90.63	1,2,4	20	22	100	42	32	96.88
6	3,4	20	22	81.82	42	39	71.79	1,2,3	20	26	100	42	32	90.63	1,2,4	20	22	95.45	42	32	71.88
7	3,4	20	22	63.64	42	32	81.25	1,2,3	20	22	81.82	42	38	89.47	1,2,4	20	22	86.36	42	39	82.05
8	3,4	20	38	73.68	42	35	80	1,2,3	20	22	86.36	42	39	89.74	1,2,4	20	38	92.11	42	38	92.11
9	3,4	20	22	72.73	42	35	82.86	1,2,3	20	34	88.24	42	32	75	1,2,4	20	22	81.82	42	32	84.38
10	3,4	20	26	84.62	42	32	84.38	1,2,3	20	38	94.74	42	38	89.47	1,2,4	20	38	86.84	42	35	85.71

Mean				**82.011**			**81.702**				**92.506**			**89.329**				**91.873**			**84.774**
		Dataset 1			**Dataset 2**				**Dataset 1**			**Dataset 2**				**Dataset 1**			**Dataset 2**		
	Prep	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**	**Prep**	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**	**Prep**	**Train**	**Test**	**Acc**	**Train**	**Test**	**Acc**
1	1,3,4	20	30	83.33	42	32	84.38	2,3,4	20	34	85.29	42	32	100	1,2,3,4	20	26	88.46	42	38	92.11
2	1,3,4	20	38	71.05	42	35	80	2,3,4	20	34	97.06	42	28	85.71	1,2,3,4	20	30	86.67	42	38	81.58
3	1,3,4	20	22	77.27	42	38	78.95	2,3,4	20	38	86.84	42	28	92.86	1,2,3,4	20	30	93.33	42	32	87.5
4	1,3,4	20	30	56.67	42	28	78.57	2,3,4	20	22	100	42	39	97.44	1,2,3,4	20	26	73.08	42	38	78.95
5	1,3,4	20	34	79.41	42	32	90.63	2,3,4	20	34	97.06	42	39	82.05	1,2,3,4	20	38	81.58	42	35	88.57
6	1,3,4	20	26	88.46	42	38	81.58	2,3,4	20	34	94.12	42	38	86.84	1,2,3,4	20	30	96.67	42	38	92.11
7	1,3,4	20	38	86.84	42	39	92.31	2,3,4	20	22	90.91	42	39	87.18	1,2,3,4	20	38	97.37	42	38	92.11
8	1,3,4	20	26	80.77	42	38	60.53	2,3,4	20	38	89.47	42	39	94.87	1,2,3,4	20	22	90.91	42	35	77.14
9	1,3,4	20	30	83.33	42	38	55.26	2,3,4	20	22	95.45	42	32	90.63	1,2,3,4	20	30	96.67	42	28	85.71
10	1,3,4	20	34	88.24	42	32	84.38	2,3,4	20	30	90	42	32	87.5	1,2,3,4	20	38	94.74	42	28	96.43
Mean				**79.537**			**78.659**				**92.62**			**90.508**				**89.948**			**87.221**

B. Cross-Validation Results for Gene Selection Methods

Gene	Prep	Test	Acc	Test	Acc	Prep	Test	Acc	Test	Acc	Prep	Test	Acc	Test	Acc	Prep	Test	Acc	Test	Acc
100	1	22	40.91	38	47.37	2	34	91.18	28	82.14	3	38	71.05	32	62.5	4	38	60.53	32	46.88
	1	34	29.41	38	42.11	2	38	92.11	35	65.71	3	34	76.47	35	51.43	4	22	54.55	32	68.75
	1	22	31.82	38	31.58	2	38	50	35	80	3	26	80.77	38	68.42	4	34	50	32	53.13
	1	26	34.62	35	48.57	2	38	81.58	32	75	3	30	86.67	39	53.85	4	26	65.38	32	53.13
	1	26	34.62	38	26.32	2	30	53.33	32	78.13	3	34	91.18	28	75	4	38	76.32	35	51.43
Average			34.28		39.19			73.64		76.2			81.23		62.24			61.36		54.66
1000	1	34	50	39	79.49	2	38	92.11	32	87.5	3	34	85.29	28	85.71	4	30	53.33	32	68.75
	1	38	47.37	39	56.41	2	26	96.15	39	87.18	3	34	85.29	38	81.58	4	30	86.67	28	64.29
	1	34	35.29	38	63.16	2	34	91.18	35	88.57	3	38	84.21	35	65.71	4	34	70.59	28	75
	1	22	50	39	41.03	2	30	96.67	28	85.71	3	22	95.45	35	82.86	4	38	65.29	38	84.21
	1	26	26.92	39	64.1	2	30	96.67	39	82.05	3	26	73.08	32	81.25	4	30	80	39	82.05
Average			41.92		60.84			94.56		86.2			84.66		79.42			71.18		74.86
3000	1	34	32.35	39	53.85	2	38	73.68	35	88.57	3	30	100	28	75	4	30	90	32	62.5
	1	22	59.09	39	71.79	2	30	96.67	30	92.31	3	38	76.32	28	85.71	4	38	81.58	28	75
	1	34	29.41	35	57.14	2	34	88.24	39	89.74	3	22	81.82	35	82.86	4	34	91.18	39	84.62
	1	22	90.91	35	71.43	2	22	86.36	35	94.29	3	38	78.95	32	78-13	4	26	84.62	35	91.43
	1	30	53.33	38	52.63	2	30	86.67	38	92.11	3	22	86.36	35	80	4	26	69.23	35	80
Average			53.02		61.37			86.32		91.4			84.69		80.89			83.32		78.71
5000	1	30	73.33	38	81.58	2	38	94.74	39	89.74	3	26	96.15	32	87.5	4	26	80.77	28	96.43
	1	22	63.64	28	60.71	2	38	92.11	39	92.31	3	30	90	39	79.49	4	34	85.29	28	89.29
	1	30	63.33	28	75	2	38	89.47	28	96.43	3	30	83.33	39	71.79	4	22	86.36	28	78.57
	1	34	67.65	39	76.92	2	22	90.91	38	92.11	3	34	88.24	38	68.42	4	34	44.12	39	71.79
	1	26	65.38	35	65.71	2	38	92.11	39	92.31	3	26	76.92	28	75	4	22	81.82	32	81.25
Average			66.67		71.98			91.87		92.58			86.93		76.44			75.67		83.47
10000	1	22	86.36	32	87.5	2	38	97.37	38	92.11	3	26	96.15	28	89.29	4	38	89.47	39	82.05
	1	38	47.37	35	77.14	2	30	96.67	32	93.75	3	26	96.15	35	80	4	26	84.62	32	84.38
	1	38	55.26	28	75	2	30	93.33	35	91.43	3	22	100	39	87.18	4	38	65.79	32	78.13
	1	30	70	39	76.92	2	38	89.47	38	94.74	3	34	88.24	32	78.13	4	26	80.77	28	92.86
	1	22	95.45	28	78.57	2	34	97.06	28	89.29	3	38	89.47	39	82.05	4	38	68.42	35	88.57
Average			70.89		79.03			94.78		92.26			94		83.33			77.81		85.2

Legend:

Dataset 1	Main Dataset (see dataset information above)
Dataset 2	Secondary Dataset (see dataset information above)
Prep	Preprocessing Method
	1 - Decimal Scale Normalization
	2 - Quantile Normalization
	3 - Z-Score Transformation
	4 - Min-Max Normalization
Train	No. of Training Samples
Test	No. of Testing Samples
Acc	Accuracy
Mean	Cross-validated accuracy

II. Source Code

A. LCCT/src/Driver.java

```
/**
 * @author Jennifer P. Cabrera
 */

import ui.UserInterface;


public class Driver {

	public static void main (String[] args) {
		javax.swing.SwingUtilities.invokeLater(new Runnable () {
			public void run () {
				new UserInterface ();
			}
		});
	}

}
```

B. LCCT/src/data/Dataset.java

```
package data;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;
import java.util.StringTokenizer;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

import libsvm.svm_model;
import util.Utils;

public class Dataset {

	private JFrame appFrame;
	private String file;
	private boolean subclassify;
	private int totalFeatures, totalSamples, totalClasses;
	private int totalTrainSamples, totalTestSamples;
	private int noOfSamplesPerClassForTesting, noOfSamplesPerClassForTraining;
	private String[] probeSet;
	private int[] selectedIndices;
```

```
private double[] rankedValues;
private double[][] trainingSet, testingSet, reducedTrainingSet,,
      trainingResults;
private ArrayList <String> sampleLabels;
private ArrayList <String> classNames;
private ArrayList <double[]> fullDataset;
      //gene expression values only, no class label
private ArrayList <ArrayList <double[]>> dataPerClass;
private ArrayList <ArrayList <Integer>> indicesPerClass;
private ArrayList <ArrayList <Integer>> trainingSamplesIndex,
      testingSamplesIndex;
private ArrayList<ArrayList<double[][]>> markerGenes;
private svm_model svmModel;
private boolean error = false;

public Dataset (String file, boolean subclassify, JFrame appFrame) {
      this.appFrame = appFrame;
      this.file = file;
      this.subclassify = subclassify;
      this.totalFeatures = this.totalSamples = this.totalClasses = 0;
      this.fullDataset = new ArrayList <double[]> ();
      this.sampleLabels = new ArrayList <String> ();
      this.classNames = new ArrayList <String> ();
      this.trainingSamplesIndex = new ArrayList <ArrayList <Integer>> ();
      this.testingSamplesIndex = new ArrayList <ArrayList <Integer>> ();

      doProcess ();
}

public int getTotalSamples () { return totalSamples; }
public int getTotalClasses () { return totalClasses; }
public int getTotalFeatures () { return totalFeatures; }
public int getTotalTrainSamples () { return totalTrainSamples; }
public int getTotalTestSamples () { return totalTestSamples; }
public String[] getProbeSet () { return probeSet; }
public double[][] getTrainingSet () { return trainingSet; }
public double[][] getTestingSet () { return testingSet; }
public ArrayList <String> getClassNames () { return classNames; }
public ArrayList<ArrayList<double[]>> getSamplesPerClass () {
      return dataPerClass;
}
public double[][] getReducedDataset () { return reducedTrainingSet; }
public boolean isSubclassified () { return subclassify; }
public int getNoOfSamplesPerClassForTraining () {
      return noOfSamplesPerClassForTraining;
}
public svm_model getSvmModel () { return svmModel; }
public int[] getSelectedGenes() { return selectedIndices; }
public double[] getRankedValues () { return rankedValues; }
public ArrayList<ArrayList<double[][]>> getMarkersPerClass () {
      return markerGenes;
}
public ArrayList <ArrayList <Integer>> getIndicesPerClass () {
      return indicesPerClass;
}
```

```
public double[][] getTrainingResults () { return trainingResults; }
public ArrayList <ArrayList <Integer>> getTrainingSamplesIndex () {
	return trainingSamplesIndex;
}
public ArrayList <ArrayList <Integer>> getTestingSamplesIndex () {
	return testingSamplesIndex;
}


public void setTrainingSet (double[][] trainSet) { trainingSet = trainSet; }
public void setTestingSet (double[][] testSet) { testingSet = testSet; }
public void setSelectedFeatures(int[] indices) { selectedIndices = indices; }
public void setReducedDataset(double[][] reducedDataset) {
	reducedTrainingSet = reducedDataset;
}
public void setRankedValues (double[] values) { rankedValues = values; }
public void setSnr(ArrayList<ArrayList<double[][]>> markersPerClass) {
	markerGenes = markersPerClass;
}
public void setSvmModel (svm_model model) { svmModel = model; }
public void setTrainingResults(double[][] results) {
	trainingResults = results;
}

private void doProcess () {
	BufferedReader br;
	try {
		br = new BufferedReader (new FileReader (file));
		probeSet = collectProbeset (br.readLine());
		totalFeatures = probeSet.length;

		String line = "";
		while ((line = br.readLine()) != null) {
			double[] sample = collectSample (line);
			fullDataset.add(sample);
		}

		totalSamples = sampleLabels.size();
		classNames = determineClassNames ();
		totalClasses = classNames.size();

		dataPerClass = getDataPerClass ();
		trainingSet = generateTrainingSet ();
		testingSet = generateTestingSet ();

		totalTrainSamples = trainingSet.length;
		totalTestSamples = testingSet.length;
	}
	catch (FileNotFoundException e) {
		JOptionPane.showMessageDialog(appFrame,
			"Dataset file not found!");
		error = true;
		//e.printStackTrace();
	}
	catch (IOException e) {
```

```
                error = true;
                //e.printStackTrace();
        }
}

private double[][] generateTestingSet () {
        Random generator = new Random ();
        noOfSamplesPerClassForTesting = (int) (generator.nextFloat() * 5) + 5;

        ArrayList <double[]> testingSetArray = new ArrayList <double[]> ();

        for (int i=0; i<dataPerClass.size(); i++) {
                if ((dataPerClass.get(i).size() -
                noOfSamplesPerClassForTraining) < noOfSamplesPerClassForTesting)
        {
        ArrayList <Integer> testSamplesIndex = new ArrayList <Integer> ();
        ArrayList <Integer> testIndices = new ArrayList <Integer> ();

        for (int j=0; j<dataPerClass.get(i).size(); j++) {
                if (!trainingSamplesIndex.get(i).contains(j)) {
                testSamplesIndex.add(j);

                testIndices.add(indicesPerClass.get(i).get(j));
                double[] sample = new double[fullDataset.get(0).length + 1];
                sample[0] = (double) i;

                for (int k=0; k<dataPerClass.get(i).get(j).length; k++) {
                        sample[k+1] = dataPerClass.get(i).get(j)[k];
                }

                testingSetArray.add(sample);
                }
        }

        testingSamplesIndex.add(testSamplesIndex);
}
else {
        ArrayList <Integer> testSamplesIndex = new ArrayList <Integer> ();
        ArrayList <Integer> testIndices = new ArrayList <Integer> ();

        int j=0;
        while (j<noOfSamplesPerClassForTesting) {
                int index = (int)(generator.nextFloat() *
                dataPerClass.get(i).size());

        if (!trainingSamplesIndex.get(i).contains(index) &&
                !testSamplesIndex.contains(index)) {
                testSamplesIndex.add(index);

                testIndices.add(indicesPerClass.get(i).get(index));
                double[] sample = new double[fullDataset.get(0).length + 1];
                sample[0] = (double) i;

                for (int k=0; k<dataPerClass.get(i).get(index).length; k++) {
                        sample[k+1] = dataPerClass.get(i).get(index)[k];
```

```
                        }
                                testingSetArray.add(sample);
                                j++;
                                }
                        }

                        testingSamplesIndex.add(testSamplesIndex);
                        }
                }

                double[][] testingSet = new double[testingSetArray.size()][];
                for (int i=0; i<testingSet.length; i++) {
                        testingSet[i] = testingSetArray.get(i);
                }

                return testingSet;
        }

        private double[][] generateTrainingSet () {
                int smallestSampleCount = dataPerClass.get(0).size();

                for (int i=0; i<dataPerClass.size(); i++) {
                        if (dataPerClass.get(i).size() < smallestSampleCount) {
                                smallestSampleCount = dataPerClass.get(i).size();
                        }
                }

                noOfSamplesPerClassForTraining = (int)((int)smallestSampleCount * 0.70);

                double[][] trainingSet = new double[noOfSamplesPerClassForTraining *
                        classNames.size()][];
                for (int i=0; i<trainingSet.length; i++) {
                        trainingSet[i] = null;
                }

                int index = 0;
                for (int i=0; i<classNames.size(); i++) {
                        ArrayList <Integer> chosenSamples =
                                getRandomSamples (noOfSamplesPerClassForTraining, i);
                        trainingSamplesIndex.add(chosenSamples);

//                      ArrayList <Integer> indices = new ArrayList <Integer> ();
//
//                      for (int j=0; j<chosenSamples.size(); j++) {
//                              indices.add(indicesPerClass.get(i).get(j));
//                      }

                        //trainingDataIndices.add(indices);

                        for (int j=0; j<chosenSamples.size(); j++) {
                                double[] sample=new double[fullDataset.get(0).length + 1];
                                sample[0] = (double) i;

                                for (int k=0; k<dataPerClass.get(i).get(j).length; k++) {
```

```
                        sample[k+1] =
                        dataPerClass.get(i).get(chosenSamples.get(j))[k];
                }

                trainingSet[index] = sample;
                index++;
            }
        }

        return trainingSet;
}

private ArrayList <Integer> getRandomSamples (int count, int classLabel) {
        Random generator = new Random ();
        ArrayList <Integer> chosenSamples = new ArrayList <Integer> ();
        int i = 0;
        while (i<count) {
            int sample = (int)(generator.nextFloat() *
                dataPerClass.get(classLabel).size());
            if (!chosenSamples.contains(sample)) {
                chosenSamples.add(sample);
                i++;
            }
        }
        return chosenSamples;
}

private ArrayList <ArrayList <double[]>> getDataPerClass () {
        ArrayList <ArrayList <double[]>> dataPerClass = new ArrayList
            <ArrayList <double[]>> ();
        indicesPerClass = new ArrayList <ArrayList <Integer>> ();
        for (int i=0; i<classNames.size(); i++) {
            dataPerClass.add(new ArrayList <double[]>());
            indicesPerClass.add(new ArrayList<Integer> ());
        }

        for (int i=0; i<sampleLabels.size(); i++) {

            dataPerClass.get(classNames.indexOf(sampleLabels.get(i))).add(ful
                lDataset.get(i));

            indicesPerClass.get(classNames.indexOf(sampleLabels.get(i))).add(
        i);
        }

        return dataPerClass;
}

private ArrayList <String> determineClassNames () {
        ArrayList <String> classNames = new ArrayList <String> ();

        for (int i=0; i<sampleLabels.size(); i++) {
            if (classNames.size() > 0) {
                if (!classNames.contains(sampleLabels.get(i)))
                    classNames.add(sampleLabels.get(i));
```

```
			} else {
				classNames.add(sampleLabels.get(i));
			}
		}

		return classNames;
	}

	private double[] collectSample (String sampleString) {
		StringTokenizer stringTokenizer = new StringTokenizer
			(sampleString, ",");
		String label = stringTokenizer.nextToken();

		if (subclassify) {
			if (label.equals("C1") || label.equals("C2") ||
				label.equals("C3")  || label.equals("C4") ||
				label.equals("CM") || label.equals("GRP1")) {
				sampleLabels.add("AD (" + label + ")");
			}
			else {
				sampleLabels.add(label);
			}
		} else {
			if (label.equals("C1") || label.equals("C2") ||
				label.equals("C3")  || label.equals("C4") ||
				label.equals("CM") || label.equals("GRP1")) {
				sampleLabels.add("AD");
			}
			else {
				sampleLabels.add(label);
			}
		}

		double[] sample = new double[totalFeatures];
		int i = 0;

		while (stringTokenizer.hasMoreTokens()) {
			sample[i] = Double.parseDouble(stringTokenizer.nextToken());
			i++;
		}

		return sample;
	}

	private String[] collectProbeset (String probes) {
		ArrayList <String> probeArray = new ArrayList <String> ();
		StringTokenizer stringTokenizer = new StringTokenizer (probes, ",");

		while (stringTokenizer.hasMoreTokens()) {
			probeArray.add(stringTokenizer.nextToken());
		}
		probeArray.remove(0);
		return Utils.convertToStringArray(probeArray);
	}
```

```
	public boolean hasError () {
		return error;
	}

}
```

C. LCCT/src/gs/MedianAbsoluteDeviation.java

```
package gs;

import java.util.ArrayList;
import java.util.Arrays;

import data.Dataset;

public class MedianAbsoluteDeviation {

	private int totalSamples, totalFeatures, count;
	private double[][] dataset, reducedDataset;
	private int[] indices;
	private double[] mads;

	public MedianAbsoluteDeviation(Dataset data, int count) {
		this.totalSamples = data.getTotalTrainSamples();
		this.totalFeatures = data.getTotalFeatures() + 1;
		this.dataset = data.getTrainingSet();
		this.count = count;

		selectGenes ();
		reducedDataset = reduceDataset ();
	}

	private double[][] reduceDataset () {
		String str = Utils.mkString (indices);
		ArrayList <double[]> reducedData = new ArrayList <double[]> ();
		double[] d;
		d = new double [totalSamples];
		for (int j=0; j<totalSamples; j++) {
			d[j] = dataset[j][0];
		}
		reducedData.add(d);
		for (int i=1; i<totalFeatures; i++) {
			if (str.contains("|"+(i-1)+"|")) {
				d = new double [totalSamples];
				for (int j=0; j<totalSamples; j++) {
					d[j] = dataset[j][i];
				}
				reducedData.add(d);
			}
		}

		reducedDataset = Utils.convertToArr(reducedData, totalSamples, count);
```

```
		//System.out.println(reducedData.size());
		return reducedDataset;
	}

	private void selectGenes () {
		mads = new double[totalFeatures];

		for (int i=1; i<totalFeatures; i++) {
			double[] feat = new double[totalSamples];
			double sum = 0;
			for (int j=0; j<totalSamples; j++) {
				feat[j] = dataset[j][i];
			}
			double median = getMedian(feat);
			for (int j=0; j<feat.length; j++) {
				sum += Math.abs(feat[j] -  median);
			}
			mads[i-1] = Math.round(((1/(double)feat.length) * sum) * 10000) /
				(double) 10000;
		}
		double[][] sortedMads = new double[totalFeatures][2];
		//init
		for (int i=0; i<mads.length; i++) {
			sortedMads[i] = new double[] {i, mads[i]};
		}
		sortedMads = Utils.sortDesc(sortedMads, 0, mads.length-1);

		indices =  new int[count];
		for (int i=0; i<indices.length; i++) {
			[i] = (int) sortedMads[i][0];
		}
	}

	private double getMedian(double[] feat) {
		double median = 0;
		Arrays.sort(feat);
		if (feat.length % 2 > 0) { median = feat[(feat.length+1)/2+1]; }
		else { median = (feat[(feat.length/2)-1] + feat[feat.length/2])/2; }
		return Math.round(median * 10000) / (double) 10000;
	}

	public double[][] getReducedDataset() { return reducedDataset;    }
	public int[] getIndices () { return indices; }
	public double[] getRankedValues () { return mads; }
}
```

D. LCCT/src/gs/StandardDeviation.java

```
package gs;

import java.util.ArrayList;
```

```
import data.Dataset;

public class StandardDeviation {

	private int totalSamples, totalFeatures, count;
	private double[][] dataset, reducedDataset;
	private int[] indices;
	private double[] sds;

	public StandardDeviation (Dataset data, int count) {
		this.totalSamples = data.getTotalTrainSamples();
		this.totalFeatures = data.getTotalFeatures() + 1;
		this.dataset = data.getTrainingSet();
		this.count = count;

		selectGenes ();
		reducedDataset = reduceDataset ();
	}

	private double[][] reduceDataset () {
		String str = Utils.mkString (indices);
		ArrayList <double[]> reducedData = new ArrayList <double[]> ();
		double[] d;
		d = new double [totalSamples];
		for (int j=0; j<totalSamples; j++) {
			d[j] = dataset[j][0];
		}
		reducedData.add(d);
		for (int i=1; i<totalFeatures; i++) {
			if (str.contains("|"+(i-1)+"|")) {
				d = new double [totalSamples];
				for (int j=0; j<totalSamples; j++) {
					d[j] = dataset[j][i];
				}
				reducedData.add(d);
			}
		}

		reducedDataset = Utils.convertToArr(reducedData, totalSamples, count);
		//System.out.println(reducedData.size());
		return reducedDataset;
	}

	/**FORMULA: sqrt(sum((x-mean(x))^2)/(n-1))**/
	public void selectGenes() {
		sds = new double[totalFeatures];

		for (int i=1; i<totalFeatures; i++) {
			double[] feat = new double[totalSamples];
			double sum = 0;
			for (int j=0; j<totalSamples; j++) {
				feat[j] = dataset[j][i];
			}
```

```
                        double mean = getMean (feat);
                        for (int j=0; j<feat.length; j++) {
                                sum += (feat[j]-mean)*(feat[j]-mean);
                        }
                        sds[i-1] = Math.round(Math.sqrt(sum/(double)(feat.length -1)) *
                                10000) / (double) 10000;
                }
                double[][] sortedSds = new double[totalFeatures][2];
                //init
                for (int i=0; i<sds.length; i++) {
                        sortedSds[i] = new double[] {i, sds[i]};
                }
                sortedSds = Utils.sortDesc(sortedSds, 0, sds.length-1);

                indices =  new int[count];
                for (int i=0; i<indices.length; i++) {
                        indices[i] = (int) sortedSds[i][0];
                }
        }

        private double getMean (double[] arr) {
                double sum = 0;
                for (int i=0; i<arr.length; i++)
                        sum += arr[i];
                return Math.round((sum/(double)arr.length) * 10000) / (double)10000;
        }

        public double[][] getReducedDataset() { return reducedDataset; }
        public int[] getIndices() {      return indices; }
        public double[] getRankedValues() { return sds; }
}
```

E. LCCT/src/gs/Utils.java

```
package gs;

import java.util.ArrayList;

public class Utils {

        public static String mkString(int[] indices) {
                String str = "";
                for (int i=0; i<indices.length; i++) {
                        str += "|" + indices[i] + "|";
                }
                return str;
        }

        public static double[][] convertToArr(ArrayList<double[]> reducedData,
                int totalSamples, int count) {
                double[][] reduced = new double[totalSamples][count+1];
                for (int i=0; i<totalSamples; i++) {
```

```
                        double[] d = new double[count+1];
                        for (int j=0; j<count+1; j++) {
                                d[j] = reducedData.get(j)[i];
                        }
                        reduced[i] = d;
                }
                return reduced;
        }

        public static double[][] sortDesc(double[][] arr, int l, int n) {
                int low = l;
                int high = n;
                if (low >= high) {  return arr; };

                int middle = (low + high) / 2;
                sortDesc (arr, low, middle);
                sortDesc (arr, middle+1, high);
                int endLow = middle;
                int startHigh = middle+1;

                while ((low <= endLow) && (startHigh <= high)) {
                        if (arr[low][1] > arr[startHigh][1]) { low++; }
                        else {
                                double[] temp = arr[startHigh];
                                for (int k=startHigh-1; k>=low; k--) {
                                        arr[k+1] = arr[k];
                                }
                                arr[low] = temp;
                                low++;
                                endLow++;
                                startHigh++;
                        }
                }
                return arr;
        }
}
```

F. LCCT/src/model/SettingsModel.java

```
package model;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import data.Dataset;

public class SettingsModel {

        private String geneFile = "";
        private boolean decnorm = true;
```

```
private boolean qnorm = false;
private boolean zscore = false;
private boolean minmax = false;
private int min = 0, max = 100;
private boolean mad = true;
private boolean sd = false;
private int gsCount = 1000;
private boolean snr = true;
private int mgCount = 100;

public void setPrep (boolean decnorm, boolean qnorm, boolean zscore,
	boolean minmax) {
	this.decnorm = decnorm;
	this.qnorm = qnorm;
	this.zscore = zscore;
	this.minmax = minmax;
}

public void setGs (boolean mad, boolean sd) {
	this.mad = mad;
	this.sd = sd;
}

public void setGsCount (int gsCount) {
	this.gsCount = gsCount;
}

public void setSnr (boolean snr) {
	this.snr = snr;
}

public void setMgCount (int mgCount) {
	this.mgCount = mgCount;
}

public void setMinMaxInterval (int min, int max) {
	this.min = min;
	this.max = max;
}

public void setGeneFile (String geneFile) {
	this.geneFile = geneFile;
}

public String getGeneFile () {
	return geneFile;
}

public boolean[] getPrep () {
	return new boolean [] { decnorm, qnorm, zscore, minmax };
}

public int[] getMinMaxInterval () {
	return new int[] { min, max };
}
```

```
	public boolean[] getGs () {
		return new boolean[] { mad, sd };
	}

	public int getGsCount () {
		return gsCount;
	}

	public boolean getSnr () {
		return snr;
	}

	public int getMgCount () {
		return mgCount;
	}

	@SuppressWarnings("resource")
	public String[] getGeneDescription (Dataset dataModel) {
		String[] genes = new String [dataModel.getProbeSet().length];

		try {
			BufferedReader bufferedReader =
			new BufferedReader (new FileReader (geneFile));
			int ctr = 0;
			String line = "";
			while ((line = bufferedReader.readLine()) != null) {
				genes[ctr] = line.substring(line.indexOf(",")+1);
				ctr++;
			}
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
		return genes;
	}

}
```

G. LCCT/src/prep/DecimalScaleNormalization.java

```
package prep;

public class DecimalScaleNormalization {

	private static int[] getDecimalPlaces (double[][] dataset, int totalSamples,
		int totalFeatures) {
		double [] maxes = new double [totalFeatures];
		int[] dec = new int [totalFeatures];
```

```
		for (int i=1; i<totalFeatures; i++) {
			double max = dataset[0][i];
			for (int j=0; j<totalSamples; j++) {
				if (dataset[j][i] > max) {
					max = dataset[j][i];
				}
			}
			maxes[i-1] = max;
		}

		for (int i=0; i<maxes.length; i++) {
			int j = 0;
			double val = Math.round((maxes[i] / Math.pow(10, j)) * 10000) /
				10000;
			while (val >=1) {
				j++;
				val = Math.round((maxes[i] / Math.pow(10, j)) * 10000) /
				10000;
			}
			dec[i] = j;
		}

		return dec;
	}

	private static double[][] performMethod(double[][] dataset, int totalSamples,
		int totalFeatures) {
		double[][] preprocessedData = new double[totalSamples][totalFeatures];
		int[] dec = getDecimalPlaces(dataset, totalSamples, totalFeatures);
		for (int row=0; row<totalSamples; row++) {
			preprocessedData[row][0] = dataset[row][0];
			for (int col=1; col<totalFeatures; col++) {
				preprocessedData[row][col] = Math.round((dataset[row][col]
				/(double) Math.pow(10, dec[col-1])) * 10000) /
				(double) 10000;
			}
		}
		return preprocessedData;
	}

	public static double[][] preprocessData(double[][] dataset, int totalSamples,
		int totalFeatures) {
		double[][] preprocessedData = performMethod (dataset,
			totalSamples,totalFeatures+1);
		return preprocessedData;
	}

}
```

H. LCCT/src/prep/MinMaxNormalization.java

```
/**
```

```
 * formula:
 * v' = ((v-min)/(max-min))(new_max - new_min) + new_min
 */

package prep;

public class MinMaxNormalization {

	public static double[][] performMethod(double[][] dataset, int totalSamples,
		int totalFeatures, int minInterval, int maxInterval) {
		double[][] processedData = new double[totalSamples][totalFeatures];
		double new_max = maxInterval, new_min=minInterval;
		double[] maxs = getMaxPerFeature (dataset, totalSamples, totalFeatures);
		double[] mins = getMinPerFeature (dataset, totalSamples, totalFeatures);

		for (int row=0; row<totalSamples; row++) {
			processedData[row][0] = dataset[row][0];
			for (int col=1; col<totalFeatures; col++) {
				processedData[row][col] = Math.round(((((dataset[row][col]
				- mins[col]) / (maxs[col]-mins[col])) * (new_max –
				new_min)) + new_min) * 10000) / (double) 10000;
			}
		}

		return processedData;
	}

	private static double[] getMinPerFeature (double[][] dataset,
		int totalSamples, int totalFeatures) {
		double[] mins = new double[totalFeatures];
		for (int i=0; i<totalFeatures; i++) { mins[i] = 0; }
		for (int row=0; row<totalSamples; row++) {
			for (int col=1; col<totalFeatures; col++) {
				if (dataset[row][col] < mins[col]) {
					mins[col] = dataset[row][col];
				};
			}
		}
		return mins;
	}

	private static double[] getMaxPerFeature (double[][] dataset,
		int totalSamples, int totalFeatures) {
		double[] maxs = new double[totalFeatures];
		for (int i=0; i<totalFeatures; i++) { maxs[i] = 0; }
		for (int row=0; row<totalSamples; row++) {
			for (int col=1; col<totalFeatures; col++) {
				if (dataset[row][col] > maxs[col]) {
					maxs[col] = dataset[row][col];
				};
			}
		}
		return maxs;
	}
```

```
	public static double[][] preprocessData(double[][] dataset, int totalSamples,
		int totalFeatures, int minInterval, int maxInterval) {
		double[][] processedData = performMethod (dataset, totalSamples,
			totalFeatures+1, minInterval, maxInterval);
		return processedData;
	}
}
```

I. LCCT/src/prep/MinMaxNormalization.java

```
/** STEPS:
 *  1. Sort each column of original matrix.
 *  2. Take average across rows.
 *  3. Substitute each value to corresponding row average.
 *  4. Unsort columns of matrix to original order.
 */

package prep;

import java.util.ArrayList;

public class QuantileNormalization {

	private static double[][] performMethod(double[][] dataset, int totalSamples,
		int totalFeatures) {
		double[][] processedData = new double[totalSamples][totalFeatures];
		ArrayList <ArrayList <double[]>> data =
			new ArrayList<ArrayList<double[]>> ();

		//init
		for (int i=0; i<totalSamples; i++) {
			ArrayList <double[]> row = new ArrayList <double[]> ();
			for (int j=0; j<totalFeatures; j++) {
				double[] rowdata = new double[2];
				rowdata[0] = i;
				rowdata[1] = dataset[i][j];
				row.add(rowdata);
			}
			data.add(row);
		}

		//transpose data to sort each column of original matrix
		ArrayList <ArrayList <double[]>> transpose =
			transposeData(data, totalSamples, totalFeatures);

		//sort
		for (int i=1; i<transpose.size(); i++) {
			transpose.set(i, mergeSort(transpose.get(i), 0,
				transpose.get(i).size()-1));
		}
```

```
		//untranspose back to original matrix
		data = untransposeData(transpose, totalSamples, totalFeatures);

		//get average across rows
		double[] average = new double[data.size()];
		for (int i=0; i<totalSamples; i++) {
			double sum = 0;
			ArrayList <double[]> row = data.get(i);
			for (int j=1; j<totalFeatures; j++) {
				double[] rowdata = row.get(j);
				sum += rowdata[1];
			}
			//average[i] = Math.round((sum / totalFeatures) * 100) / 100;
			average[i] = Math.round((sum / totalFeatures) * 10000) /
				(double)10000;
		}

		//substitute each value with corresponding row average
		for (int i=0; i<totalSamples; i++) {
			ArrayList <double[]> row = data.get(i);
			for (int j=1; j<totalFeatures; j++) {
				double[] rowdata = row.get(j);
				rowdata[1] = average[i];
				row.set(j, rowdata);
			}
		}

		//unsort columns of matrix to orig order
		processedData = unsortMatrix (data, totalSamples, totalFeatures);
		return processedData;
	}

	private static double[][] unsortMatrix(ArrayList<ArrayList<double[]>> data,
		int totalSamples, int totalFeatures) {
		double[][] processedData = new double[totalSamples][totalFeatures];
		for (int i=0; i<totalSamples; i++) {
			ArrayList <double[]> row = data.get(i);
			for (int j=0; j<totalFeatures; j++) {
				double[] rowdata = row.get(j);
				processedData[(int) rowdata[0]][j] = rowdata[1];
			}
		}
		return processedData;
	}

	private static ArrayList <ArrayList <double[]>> untransposeData (
		ArrayList <ArrayList <double[]>> data, int totalSamples,
			int totalFeatures) {
		ArrayList <ArrayList <double[]>> transpose =
			new ArrayList<ArrayList<double[]>> ();
		for (int i=0; i<totalSamples; i++) {
			ArrayList <double[]> transposeRow = new ArrayList <double[]> ();
			for (int j=0; j<totalFeatures; j++) {
				ArrayList <double[]> row = data.get(j);
```

```
                        double[] rowdata = row.get(i);
                        transposeRow.add(rowdata);
                }
                transpose.add(transposeRow);
        }
        return transpose;
}

private static ArrayList <ArrayList <double[]>> transposeData (
        ArrayList <ArrayList <double[]>> data, int totalSamples,
                int totalFeatures) {
        ArrayList <ArrayList <double[]>> transpose =
                new ArrayList<ArrayList<double[]>> ();
        for (int i=0; i<totalFeatures; i++) {
                ArrayList <double[]> transposeRow = new ArrayList <double[]> ();
                for (int j=0; j<totalSamples; j++) {
                        ArrayList <double[]> row = data.get(j);
                        double[] rowdata = row.get(i);
                        transposeRow.add(rowdata);
                }
                transpose.add(transposeRow);
        }
        return transpose;
}

//mergesort
private static ArrayList<double[]> mergeSort (ArrayList<double[]> row, int l,
        int h) {
        int low = l;
        int high = h;
        if (low >= high) { return row; };

        int middle = (low + high) / 2;
        mergeSort(row, low, middle);
        mergeSort(row, middle+1, high);

        int endLow = middle;
        int startHigh = middle+1;
        while ((low <= endLow) && (startHigh <= high)) {
                if (row.get(low)[1] < row.get(startHigh)[1]) { low++; }
                else {
                        double[] temp = row.get(startHigh);
                        for (int k=startHigh-1; k>=low; k--) {
                                row.set(k+1, row.get(k));
                        }
                        row.set(low, temp);
                        low++;
                        endLow++;
                        startHigh++;
                }
        }
        return row;
}

public static double[][] preprocessData (double[][] dataset, int totalSamples,
```

```
	int totalFeatures) {
		double[][] preprocessedData = performMethod (dataset, totalSamples,
			totalFeatures+1);
		return preprocessedData;
	}

}
```

J. LCCT/src/prep/MinMaxNormalization.java

```
/**
 * FORMULA:
 * z-Score = (x - mean) / stddev
 */

package prep;

public class ZScoreTransformation {

	public static double[][] performMethod(double[][] dataset, int totalSamples,
		int totalFeatures) {
		double [][]processedData = new double[totalSamples][totalFeatures];
		double[] meanIntensitiesPerFeature =
			getMeanIntensityPerFeature (dataset, totalSamples,
			totalFeatures);
		double[] stdDeviationPerFeature =
			getStdDeviationPerFeature (meanIntensitiesPerFeature, dataset,
			totalSamples, totalFeatures);

		for (int row=0; row<totalSamples; row++) {
			processedData[row][0] = dataset[row][0];
			for (int col=1; col<totalFeatures; col++) {
				processedData[row][col] = Math.round(((dataset[row][col] -
				 meanIntensitiesPerFeature[col]) /
				stdDeviationPerFeature[col])*10000) / (double)10000;
			}
		}
		return processedData;

	}

	/**FORMULA: sqrt(sum((x-mean(x))^2)/(n-1))**/
	private static double[] getStdDeviationPerFeature (double[] meanPerFeature,
		double[][] dataset, int totalSamples, int totalFeatures) {
		double[] stdDev = new double[totalFeatures];
		for (int i=0; i<totalFeatures; i++) { stdDev[i] = 0; }

		for (int row=0; row<totalSamples; row++) {
			for (int col=1; col<totalFeatures; col++) { //first col = class
				stdDev[col] += (double) (dataset[row][col] -
				 meanPerFeature[col]) * (double)(dataset[row][col] -
				 meanPerFeature[col]);
```

```
                }
            }
            for (int i=0; i<totalFeatures; i++) { stdDev[i] = Math.sqrt(stdDev[i] /
                (double) (totalSamples - 1)); }

            return stdDev;
        }

        private static double[] getMeanIntensityPerFeature (double[][] dataset, int
            totalSamples, int totalFeatures) {
            double[] means = new double [totalFeatures];
            for (int i=0; i<totalFeatures; i++) { means[i] = 0; }

            for (int row=0; row<totalSamples; row++) {
                for (int col=1; col<totalFeatures; col++) {
                // first col = class
                        means[col] += (double) dataset[row][col];
                /*Summation of features*/
                }
            }
            for (int i=1; i<totalFeatures; i++) {
                means[i] = (double)means[i] / (double)totalSamples;
            }

            return means;
        }

        public static double[][] preprocessData(double[][] dataset, int totalSamples,
            int totalFeatures) {
            double [][]processedData = performMethod (dataset, totalSamples,
                totalFeatures+1);
            return processedData;
        }
}
```

K. LCCT/src/proc/LcctProcessor.java

```
package proc;

import java.util.ArrayList;
import java.util.concurrent.CancellationException;
import java.util.concurrent.ExecutionException;

import gs.MedianAbsoluteDeviation;
import gs.StandardDeviation;
import gs.Utils;

import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.SwingWorker;
```

```
import prep.DecimalScaleNormalization;
import prep.MinMaxNormalization;
import prep.QuantileNormalization;
import prep.ZScoreTransformation;
import snr.SignalToNoiseRatio;
import svm.SvmEvaluator;
import svm.SvmParameters;
import svm.SvmTrainer;
import ui.ResultDatasetTab;
import ui.ResultMarkerGenesTab;
import ui.ResultParametersTab;
import ui.ResultPredictionTab;
import ui.ResultSelectedGenesTab;
import ui.ResultStatsTab;
import libsvm.svm_parameter;
import model.SettingsModel;
import data.Dataset;

public class LcctProcessor extends SwingWorker <JTabbedPane, Void>{

	private JTabbedPane resultTabbedPane;
	private JFrame appFrame;
	private Dataset dataModel;
	private SettingsModel settings;
	private svm_parameter svmParams;
	private ArrayList <JTabbedPane> resultTabbedPanes;

	public LcctProcessor(Dataset dataModel, SettingsModel settings,
		JTabbedPane resultTabbedPane, JFrame appFrame/*,
		ArrayList <JTabbedPane> resultTabbedPanes*/) {
		this.appFrame = appFrame;
		this.dataModel = dataModel;
		this.settings = settings;
		this.resultTabbedPane = resultTabbedPane;

		resultTabbedPane.removeAll();
		resultTabbedPane.addTab ("Dataset",
		new ResultDatasetTab (dataModel));
		resultTabbedPane.addTab ("Parameters", new JScrollPane (null,
		JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
		JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
		resultTabbedPane.addTab ("Selected Genes", new JScrollPane (null,
		JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
		JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
		resultTabbedPane.addTab ("Marker Genes", new JScrollPane (null,
		JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
		JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
		resultTabbedPane.addTab ("Prediction", new JScrollPane (null,
		JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
		JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
		resultTabbedPane.addTab("Performance", new JScrollPane (null,
		JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
		JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
```

```
            for (int i=1; i<resultTabbedPane.getTabCount(); i++) {
                  resultTabbedPane.setEnabledAt(i, false);
            }

            SvmParameters params = new SvmParameters();
            svmParams = params.getSvmParameters();
      }

      @Override
      protected JTabbedPane doInBackground() throws Exception {
            resultTabbedPane.setEnabled(true);
            setProgress(5);

//          resultTabbedPane.setComponentAt(0, new ResultDatasetTab (dataModel));
//          resultTabbedPane.setEnabledAt(0, true);
//          resultTabbedPane.setSelectedIndex(0);

            resultTabbedPane.setComponentAt(1, new JScrollPane (
            new ResultParametersTab (svmParams, settings),
            JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
            JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
            resultTabbedPane.setEnabledAt(1, true);
            resultTabbedPane.setSelectedIndex(1);

            setProgress (15);

            if (isCancelled ()) {
                  JOptionPane.showMessageDialog(appFrame,
                              "Process cancelled!",
                              "Warning",
                              JOptionPane.WARNING_MESSAGE);
            } else {
                  double[][] trainingSet = dataModel.getTrainingSet();
                  double[][] testingSet = dataModel.getTestingSet();

                  if (settings.getPrep()[0]) {
                        trainingSet =
                        DecimalScaleNormalization.preprocessData(trainingSet,
                        dataModel.getTotalTrainSamples(),
                        dataModel.getTotalFeatures());
                        testingSet =
                        DecimalScaleNormalization.preprocessData(testingSet,
                        dataModel.getTotalTestSamples(),
                        dataModel.getTotalFeatures());
                  }
                  if (settings.getPrep()[1]) {
                        trainingSet =
                        QuantileNormalization.preprocessData(trainingSet,
                        dataModel.getTotalTrainSamples(),
                        dataModel.getTotalFeatures());
                        testingSet =
                        QuantileNormalization.preprocessData(testingSet,
                        dataModel.getTotalTestSamples(),
                        dataModel.getTotalFeatures());
                  }
```

```
if (settings.getPrep()[2]) {
	trainingSet =
	ZScoreTransformation.preprocessData(trainingSet,
	dataModel.getTotalTrainSamples(),
	dataModel.getTotalFeatures());
	testingSet =
	ZScoreTransformation.preprocessData(testingSet,
	dataModel.getTotalTestSamples(),
	dataModel.getTotalFeatures());
}
if (settings.getPrep()[3]) {
	trainingSet =
	MinMaxNormalization.preprocessData(trainingSet,
	dataModel.getTotalTrainSamples(),
	dataModel.getTotalFeatures(),
	settings.getMinMaxInterval()[0],
	settings.getMinMaxInterval()[1]);
	testingSet =
	MinMaxNormalization.preprocessData(testingSet,
	dataModel.getTotalTestSamples(),
	dataModel.getTotalFeatures(),
	settings.getMinMaxInterval()[0],
	settings.getMinMaxInterval()[1]);
}
dataModel.setTrainingSet(trainingSet);
dataModel.setTestingSet(testingSet);

setProgress (30);

if (settings.getGs()[0]) {
	MedianAbsoluteDeviation mad = new
	MedianAbsoluteDeviation (dataModel,
settings.getGsCount());
	dataModel.setSelectedFeatures(mad.getIndices());
	dataModel.setReducedDataset(mad.getReducedDataset());
	dataModel.setRankedValues(mad.getRankedValues());
} else
if (settings.getGs()[1]) {
	StandardDeviation sd =
	new StandardDeviation (dataModel, settings.getGsCount());
	dataModel.setSelectedFeatures(sd.getIndices());
	dataModel.setReducedDataset(sd.getReducedDataset());
	dataModel.setRankedValues(sd.getRankedValues());
}

setProgress (50);
resultTabbedPane.setComponentAt(2, new JScrollPane (
new ResultSelectedGenesTab (appFrame, settings, dataModel),
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
resultTabbedPane.setEnabledAt(2, true);
resultTabbedPane.setSelectedIndex(2);

if (settings.getSnr()) {
	new SignalToNoiseRatio (dataModel, settings.getGsCount(),
```

```
            settings.getMgCount());
        }

        setProgress (60);
        resultTabbedPane.setComponentAt(3, new JScrollPane (
        new ResultMarkerGenesTab (settings, dataModel),
        JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
        resultTabbedPane.setEnabledAt(3, true);
        resultTabbedPane.setSelectedIndex(3);

        trainingSet = dataModel.getReducedDataset();

        dataModel.setSvmModel(SvmTrainer.getSvmModel(svmParams,
        dataModel));

        double[][] trainingResults =
        new double[dataModel.getTestingSet().length][];

        for (int j=0; j<dataModel.getTestingSet().length; j++) {
            if (isCancelled()) {
                JOptionPane.showMessageDialog(appFrame,
                            "Process cancelled!",
                            "Warning",
                            JOptionPane.WARNING_MESSAGE);
            } else {
                double[] d =
                reduceDataset(dataModel.getTestingSet()[j],
                dataModel.getSelectedGenes());
                trainingResults[j] = SvmEvaluator.evaluate(d,
                dataModel.getSvmModel(),
                dataModel.getTotalClasses());
            }
            dataModel.setTrainingResults (trainingResults);
        }
}

System.out.println("Train: " + dataModel.getTrainingSet().length);
System.out.println("Test: " + dataModel.getTestingSet().length);

resultTabbedPane.setComponentAt(4, new JScrollPane (
new ResultPredictionTab (dataModel),
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
resultTabbedPane.setEnabledAt(4, true);
resultTabbedPane.setSelectedIndex(4);
setProgress (80);

resultTabbedPane.setComponentAt(5,
new JScrollPane (new ResultStatsTab (dataModel),
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
resultTabbedPane.setEnabledAt(5, true);
resultTabbedPane.setSelectedIndex(5);
setProgress(100);
```

```
            return resultTabbedPane;
        }

        private double[] reduceDataset (double[] data, int[] indices) {
            String str = Utils.mkString (indices);
            double[] d = new double[settings.getGsCount()+1];
            int index = 0;
            d[index] = data[0];
            index++;
            for (int i=1; i<data.length; i++) {
                if (str.contains("|"+(i-1)+"|")) {
                    d[index] = data[i];
                    index++;
                }
            }
            return d;
        }


        protected void done () {
            try {
                JTabbedPane resultPane = get ();
                resultTabbedPanes.add(resultPane);
            } catch (InterruptedException e) {
                JOptionPane.showMessageDialog(appFrame,
                        "Process interrupted!",
                        "Warning",
                        JOptionPane.WARNING_MESSAGE);
                e.printStackTrace();
                return;
            } catch (ExecutionException e) {
                JOptionPane.showMessageDialog(appFrame,
                        "Execution Exception!",
                        "Warning",
                        JOptionPane.WARNING_MESSAGE);
                e.printStackTrace();
                return;
            } catch (CancellationException e) {
                JOptionPane.showMessageDialog(appFrame,
                        "Process cancelled!",
                        "Warning",
                        JOptionPane.WARNING_MESSAGE);
                e.printStackTrace();
                return;
            }
        }

}
```

L. LCCT/src/snr/SignalToNoiseRatio.java

```
package snr;

import java.util.ArrayList;

import data.Dataset;

public class SignalToNoiseRatio {

	private Dataset dataset;
	private double[][] datasetData;
	private int gscount, mgcount;

	public SignalToNoiseRatio (Dataset dataset, int gscount, int mgcount) {
		this.dataset = dataset;
		this.datasetData = dataset.getReducedDataset();
		this.gscount = gscount;
		this.mgcount = mgcount;

		getSnrs ();
	}

	private void getSnrs () {
		double[][] meansPerGenePerClass = getMeansPerGenePerClass ();
		double[][] stddevPerGenePerClass =
			getStdDevPerGenePerClass (meansPerGenePerClass);

		ArrayList <ArrayList<double[][]>> markersPerClass =
			new ArrayList <ArrayList <double[][]>> ();

		for (int i=0; i<dataset.getTotalClasses(); i++) {
			ArrayList <double[][]> classSnr = new ArrayList <double[][]> ();
			for (int j=0; j<dataset.getTotalClasses(); j++) {
				if (i != j) {
					double[][] sampleSnr = new double[gscount][2];
					for (int k=0; k<gscount; k++) {
						double snr [] = new double[2];
						snr[0] = k;
						snr[1] =
Math.round((meansPerGenePerClass[i][k] - meansPerGenePerClass[j][k]) /
(stddevPerGenePerClass[i][k] - stddevPerGenePerClass[j][k]) * (double)10000) /
(double)10000;
						sampleSnr[k] = snr;
					}
					sampleSnr = sortDesc(sampleSnr, 0, sampleSnr.length-
1);

					double[][] temp = new double[mgcount][2];
					for (int k=0; k<temp.length; k++) {
						temp[k] = sampleSnr[k];
					}
					classSnr.add(temp);
				}
			}
			markersPerClass.add(classSnr);
		}
		dataset.setSnr (markersPerClass);
```

```
        //System.out.println("/**********************/" +
        markersPerClass.size());
}

public double[][] sortDesc(double[][] arr, int l, int n) {
        int low = l;
        int high = n;
        if (low >= high) {  return arr; };

        int middle = (low + high) / 2;
        sortDesc (arr, low, middle);
        sortDesc (arr, middle+1, high);
        int endLow = middle;
        int startHigh = middle+1;

        while ((low <= endLow) && (startHigh <= high)) {
                if (arr[low][1] > arr[startHigh][1]) { low++; }
                else {
                        double[] temp = arr[startHigh];
                        for (int k=startHigh-1; k>=low; k--) {
                                arr[k+1] = arr[k];
                        }
                        arr[low] = temp;
                        low++;
                        endLow++;
                        startHigh++;
                }
        }
        return arr;
}

private double[][] getStdDevPerGenePerClass (double[][] means) {
        double[][] stddevs = new double[dataset.getTotalClasses()][gscount];


        for (int i=0; i<dataset.getTotalClasses(); i++) {
                double[] stddev = new double[gscount];
                for (int j=0; j<stddev.length; j++) {
                        stddev[j] = 0;
                }
                for (int j=0; j<datasetData.length; j++) {
                        if (i == datasetData[j][0]) {
                                for (int k=0; k<gscount; k++) {
                                        stddev[k] += (datasetData[j][k+1] -
                                        means[i][k])*(datasetData[j][k+1] -
                                        means[i][k]);
                                }
                        }
                }

                for (int j=0; j<stddev.length; j++) {
                        stddev[j] = Math.round(Math.sqrt((stddev[j] /
                        (double)(dataset.getNoOfSamplesPerClassForTraining() -
                        1))) * (double)10000) / (double)10000;
                }
```

```
                stddevs[i] = stddev;
            }


            return stddevs;
        }

        private double[][] getMeansPerGenePerClass () {
            double[][] means = new double[dataset.getTotalClasses()][gscount];

            for (int i=0; i<dataset.getTotalClasses(); i++) {
                double[] mean = new double[gscount];
                for (int j=0; j<mean.length; j++) {
                    mean[j] = 0;
                }
                for (int j=0; j<datasetData.length; j++) {
                    if (i == datasetData[j][0]) {
                        for (int k=0; k<gscount; k++) {
                            mean[k] += datasetData[j][k+1];
                        }
                    }
                }

                for (int j=0; j<mean.length; j++) {
                    mean[j] = Math.round((mean[j] /
                    (double)dataset.getNoOfSamplesPerClassForTraining()) *
                    (double)10000)  / (double)10000;
                }

                means[i] = mean;
            }
            return means;
        }


}
```

M. LCCT/src/svm/SvmEvaluator.java

```
package svm;


import libsvm.svm;
import libsvm.svm_model;
import libsvm.svm_node;

public class SvmEvaluator {

        public static double[] evaluate (double[] features, svm_model model,
            int totalClasses) {
            //System.out.println("features :" + features.length);
```

```
		svm_node[] nodes = new svm_node[features.length-1];
		for (int i=1; i<features.length; i++) {
			svm_node node = new svm_node();
			node.index = i;
			node.value = features[i];

			nodes[i-1] = node;
		}

		int[] labels = new int[totalClasses];
		svm.svm_get_labels(model, labels);

		double[] prob_estimates = new double[totalClasses];
		double v = svm.svm_predict_probability(model, nodes, prob_estimates);

		double[] result = new double[totalClasses+2];
		result[0] = features[0];
		result[1] = v;

		//if (features[0] == v) { hits++; }

		for (int i=0; i<totalClasses; i++) {

			result[i+2] = prob_estimates[i];
		}
		return result;
	}
}
```

N. LCCT/src/svm/SvmParameters.java

```
package svm;


import libsvm.*;

public class SvmParameters {

	public static svm_parameter param;

	public SvmParameters() {
		param = new svm_parameter ();
		//default values

		//param.svm_type = svm_parameter.C_SVC;
		//param.kernel_type = svm_parameter.RBF;
		//param.degree = 3;
		//param.gamma = 0;
		//param.coef0 = 0;
		//param.nu = 0.5;
		//param.cache_size = 20000;
```

```
		//param.C = 1;
		//param.eps = 1e-3;
		//param.p = 0.1;
		//param.shrinking = 1;
		//param.probability = 0;
		//param.nr_weight = 0;
		//param.weight_label = new int[0];
		//param.weight = new double[0];


		// defaults: linear kernel
		param.probability = 1;
		param.gamma = 0.5;
		param.nu = 0.5;
		param.C = 1;
		param.svm_type = svm_parameter.C_SVC;
		param.kernel_type = svm_parameter.LINEAR;
		param.cache_size = 20000;
		param.eps = 0.001;
	}

	public svm_parameter getSvmParameters () {
		return param;
	}
}
```

O. LCCT/src/svm/SvmParameters.java

```
package svm;
import data.Dataset;
import libsvm.*;

public class SvmTrainer {

	private static svm_model trainSystem (svm_parameter param, double[][] dataset,
		int totalSamples) {
		svm_problem prob = new svm_problem ();
		prob.l = totalSamples;
		prob.y = new double[prob.l];

		if (param.gamma == 0) { param.gamma = 1; }
		prob.x = new svm_node[prob.l][];
		for (int i=0; i<totalSamples; i++) {
			double[] features = dataset[i];
			prob.x[i] = new svm_node[features.length-1];
			for (int j=1; j<features.length; j++) {
				svm_node node = new svm_node();
				node.index = j;
				node.value = features[j];
				prob.x[i][j-1] = node;
			}
			prob.y[i] = features[0];
```

```
            }

            return svm.svm_train(prob, param);
      }

      public static svm_model getSvmModel(svm_parameter param, Dataset dataset) {
            svm_model model = trainSystem (param, dataset.getReducedDataset(),
            dataset.getTotalTrainSamples());
            return model;
      }
}
```

P. LCCT/src/ui/HelpFrame.java

```
package ui;

import java.awt.Color;
import java.awt.Font;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;

@SuppressWarnings("serial")
public class HelpFrame extends JFrame {

      private final JFrame helpFrame;
      private UIManager.LookAndFeelInfo[] looks;
      private JLayeredPane mainlpane;

      public HelpFrame () {
            helpFrame = new JFrame ("LCCT Tutorial and Guide");
            helpFrame.setSize(600, 400);

            looks = UIManager.getInstalledLookAndFeels();
            changeLookAndFeel (3);

            looks = UIManager.getInstalledLookAndFeels();
            changeLookAndFeel (3);

            mainlpane = new JLayeredPane ();
            mainlpane.setBounds(0,25,756,100);

            JPanel bg = new JPanel ();
            bg.setBackground(Color.decode("#87CEFA"));
```

```
            bg.setBounds(0, 0, 600, 400);
            mainlpane.add(bg);

            JPanel panel = new JPanel ();
            panel.setBackground(Color.WHITE);
            panel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            panel.setBounds(5, 5, 583, 360);
            mainlpane.add(panel, 1, new Integer (0));

            JLabel label = new JLabel ("LCCT: Lung Cancer Classification Tool");
            label.setFont(new Font ("Tahoma", Font.ITALIC|Font.BOLD, 15));
            label.setBounds(20, 15, 300, 50);
            mainlpane.add(label, 2, new Integer(1));

            JLabel infoLabel1 = new JLabel ("<html><align=justify>Welcome to LCCS -
"                       + "a tool for classifying benign and malignant lung cancer
and "                   + "predicting its corresponding subclasses based on
microarray data. "      + "This tool uses preprocessing and gene selection methods
to "                    + "clean data and Support Vector Machines (SVM) to
generate "              + "learning models for use in the prediction.</html>");
            infoLabel1.setBounds(30, 40, 300, 200);
            mainlpane.add(infoLabel1, 2, new Integer(1));

            JLabel infoLabel2 = new JLabel ("<html><align=justify>This window will
walk you "          + "through the steps into using this tool. </html>");
            infoLabel2.setBounds(30, 110, 300, 200);
            mainlpane.add(infoLabel2, 2, new Integer(1));

            JLabel imgLabel = new JLabel ();
            imgLabel.setIcon(new ImageIcon ("img/oligo.jpg"));
            imgLabel.setBounds(380, 100, 148, 131);
            mainlpane.add(imgLabel, 2, new Integer(1));

            JButton prevBtn = new JButton ("<<");
            prevBtn.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            prevBtn.setBackground(Color.decode("#87CEFA"));
            prevBtn.setBounds(5, 340, 100, 25);
            mainlpane.add(prevBtn, 2, new Integer(1));
            JButton nextBtn = new JButton (">>");
            nextBtn.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            nextBtn.setBackground(Color.decode("#87CEFA"));
            nextBtn.setBounds(488, 340, 100, 25);
            mainlpane.add(nextBtn, 2, new Integer(1));

            helpFrame.add(mainlpane);
            helpFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            helpFrame.setVisible(true);
            helpFrame.setResizable(false);
      }

      private void changeLookAndFeel (int value) {
            try {
                  UIManager.setLookAndFeel(looks[value].getClassName());
                  SwingUtilities.updateComponentTreeUI(helpFrame);
            } catch (Exception e) {
```

```
                    e.printStackTrace();
                }
        }
}
```

Q. LCCT/src/ui/MatrixPanel.java

```
package ui;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;

import javax.swing.JPanel;

@SuppressWarnings("serial")
public class MatrixPanel extends JPanel {

        private int width, height;
        private int ROWS, COLS, PAD = 10;
        private int[][] matrix;
        private ArrayList <String> classNames;

        public MatrixPanel (int[][] matrix, ArrayList<String> classNames, int width,
                int height, int ROWS, int COLS) {
                this.width = width;
                this.height = height;
                this.ROWS = ROWS;
                this.COLS = COLS;
                this.matrix = matrix;
                this.classNames = classNames;

                setBackground (Color.WHITE);
        }

        public void paintComponent (Graphics g) {
                super.paintComponent(g);
                Graphics2D g2 = (Graphics2D) g;
                double w = width;//getWidth();
                double h = height;//getHeight();
                double xInc = (w - 2*PAD)/COLS;
                double yInc = (h - 2*PAD)/ROWS;
                for(int j = 0; j < ROWS; j++) {
                double y = PAD + j*yInc;
                for(int k = 0; k < COLS; k++) {
                if (j==0 && k==0) {
                } else if (j==k) {
                        double x = PAD + k*xInc;
                        g2.setPaint(Color.yellow);
```

```
                        g2.fill(new Rectangle2D.Double(x, y, xInc, yInc));
                        g2.setPaint(Color.lightGray);
                        g2.draw(new Rectangle2D.Double(x, y, xInc, yInc));
                        g2.setPaint(Color.darkGray);
                              g2.setFont(new Font ("Sans Serif", Font.PLAIN, 12));
                              g2.drawString(Integer.toString(matrix[j-1][k-1]),
                              (int)(x+(xInc/2)-1) , (int)(y+yInc));
                  } else if (j==0) {
                        double x = PAD + k*xInc;
                        g2.setPaint(Color.pink);
                        g2.fill(new Rectangle2D.Double(x, y, xInc, yInc));
                        g2.setPaint(Color.lightGray);
                        g2.draw(new Rectangle2D.Double(x, y, xInc, yInc));
                        g2.setPaint(Color.DARK_GRAY);
                        g2.setFont(new Font ("Sans Serif", Font.PLAIN, 12));
                              g2.drawString(classNames.get(k-1), (int)(x) ,
                              (int)(y+yInc));
                  } else if (k==0) {
                        double x = PAD + k*xInc;
                        g2.setPaint(Color.cyan);
                        g2.fill(new Rectangle2D.Double(x, y, xInc, yInc));
                        g2.setPaint(Color.lightGray);
                        g2.draw(new Rectangle2D.Double(x, y, xInc, yInc));
                        g2.setPaint(Color.DARK_GRAY);
                        g2.setFont(new Font ("Sans Serif", Font.PLAIN, 12));
                              g2.drawString(classNames.get(j-1), (int)(x) ,
                              (int)(y+yInc));
                  } else {
                        double x = PAD + k*xInc;
                        g2.setPaint(Color.white);
                        g2.fill(new Rectangle2D.Double(x, y, xInc, yInc));
                        g2.setPaint(Color.lightGray);
                        g2.draw(new Rectangle2D.Double(x, y, xInc, yInc));
                        g2.setPaint(Color.darkGray);
                              g2.setFont(new Font ("Sans Serif", Font.PLAIN, 12));
                              g2.drawString(Integer.toString(matrix[j-1][k-1]),
                              (int)(x+(xInc/2)-1) , (int)(y+yInc));
                        }
                  }
            }

      }
}
```

R. LCCT/src/ui/ResultDatasetTab.java

```
package ui;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridBagConstraints;
```

```
import java.awt.GridBagLayout;
import java.util.ArrayList;

import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableCellRenderer;

import data.Dataset;

@SuppressWarnings("serial")
public class ResultDatasetTab extends JPanel {

	private Dataset dataModel;

	public ResultDatasetTab () {
		setBackground (Color.white);
		GridBagLayout gridbag = new GridBagLayout ();
		GridBagConstraints c = new GridBagConstraints ();
		setLayout(gridbag);

		c.weightx = 0;
		c.weighty = 0;
		c.fill = GridBagConstraints.BOTH;;
		c.gridx = 0;
		c.gridy = 0;
		c.gridwidth = 1;
		c.gridheight = 1;
		JPanel trainSetPane = createEmptyInfoPane();
		trainSetPane.setBackground(Color.WHITE);
		trainSetPane.setBorder(BorderFactory.createTitledBorder(
			"Dataset Information"));
		gridbag.setConstraints(trainSetPane, c);
		add (trainSetPane);

		c.weightx = 1000;
		c.weighty = 1000;
		c.fill = GridBagConstraints.BOTH;
		c.gridx = 0;
		c.gridy = 1;
		c.gridwidth = 1;
		c.gridheight = 1;
		JScrollPane infoPane = new JScrollPane (createEmptyTrainSetPane(),
		JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
		JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
		infoPane.setBorder(BorderFactory.createTitledBorder(
		"Generated Training Set"));
		gridbag.setConstraints(infoPane, c);
		add (infoPane);
	}

	public ResultDatasetTab (Dataset dataModel) {
```

```
        this.dataModel = dataModel;

        GridBagLayout gridbag = new GridBagLayout ();
        GridBagConstraints c = new GridBagConstraints ();
        setLayout(gridbag);

        c.weightx = 0;
        c.weighty = 0;
        c.fill = GridBagConstraints.BOTH;;
        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        JPanel trainSetPane = createInfoPane();
        trainSetPane.setBackground(Color.WHITE);
        trainSetPane.setBorder(BorderFactory.createTitledBorder(
        "Dataset Information"));
        gridbag.setConstraints(trainSetPane, c);
        add (trainSetPane);

        c.weightx = 1000;
        c.weighty = 1000;
        c.fill = GridBagConstraints.BOTH;
        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        //JPanel infoPanel = new JPanel ();
        JScrollPane infoPane = new JScrollPane (createTrainSetPane(),
        JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
        JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        //infoPanel.add(infoPane);
        infoPane.setBorder(BorderFactory.createTitledBorder(
        "Generated Training Set"));
        gridbag.setConstraints(infoPane, c);
        add (infoPane);
}

private JPanel createEmptyInfoPane () {
        JPanel panel = new JPanel ();
        panel.setBackground(Color.WHITE);

        return panel;
}

private JPanel createInfoPane () {
        JPanel panel = new JPanel ();
        panel.setLayout(new FlowLayout (FlowLayout.LEFT));

        Object[] columnNames = { 1, 2};
        Object[][] rowData1 = {
                        {"Total Samples: ", dataModel.getTotalSamples()},
                        {"Total Genes: ", dataModel.getTotalFeatures()},
                        {"Total Classes: ", dataModel.getTotalClasses()},
                        {"Training Set: ", dataModel.getTotalTrainSamples() +
```

```
                        " samples"},
                {"Testing Set: ", dataModel.getTotalTestSamples() +
                        " samples"},
                {"Subclassified: ", dataModel.isSubclassified()}
};

JTable table = new JTable (rowData1, columnNames);
table.setPreferredSize(new Dimension (190, 120));
table.setOpaque(false);
table.setTableHeader(null);
table.setShowGrid(false);
table.getColumnModel().getColumn(0).setPreferredWidth(100);
panel.add(table);

ArrayList <Object[]> rowDataArr2 = new ArrayList <Object[]> ();
ArrayList <Object[]> rowDataArr1 = new ArrayList <Object[]> ();
int mainClassCount = 0;
String mainClass = "";
rowDataArr1.add(new Object[] {"Main Class", "Samples"});
for (int i=0; i<dataModel.getSamplesPerClass().size(); i++) {
       if (dataModel.getClassNames().get(i).contains("(")) {
              mainClassCount +=
                     dataModel.getSamplesPerClass().get(i).size();
              mainClass = dataModel.getClassNames().get(i).substring(0,
                     dataModel.getClassNames().get(i).indexOf("("));
              rowDataArr2.add(new Object [] {
                     dataModel.getClassNames().get(i).substring(dataModel
                     .getClassNames().get(i).indexOf("(")+1,
                     dataModel.getClassNames().get(i).indexOf(")")),
                     dataModel.getSamplesPerClass().get(i).size()});
       }
       else {
              rowDataArr1.add(new Object [] {
              dataModel.getClassNames().get(i),
              dataModel.getSamplesPerClass().get(i).size()});
       }
}
if (mainClassCount != 0)
       rowDataArr1.add(new Object [] {mainClass, mainClassCount});

Object[][] rowData2 = new Object [rowDataArr1.size()][];
for (int i=0; i<rowDataArr1.size(); i++) {
       rowData2[i] = rowDataArr1.get(i);
}

JTable classTable = new JTable (rowData2, columnNames);
classTable.setOpaque(false);
classTable.setPreferredSize(new Dimension (150, 120));
classTable.setTableHeader(null);
classTable.setShowGrid(false);
classTable.getColumnModel().getColumn(0).setPreferredWidth(100);
panel.add(classTable);

if (!mainClass.equals("") && rowDataArr2.size()!=0) {
       Object[][] rowData3 = new Object [rowDataArr2.size()+1][];
```

```
                rowData3[0] = new Object [] {mainClass + " Subclass", "Samples"};
                for (int i=0; i<rowDataArr2.size(); i++) {
                        rowData3[i+1] = rowDataArr2.get(i);
                }

                JTable subclassTable = new JTable (rowData3, columnNames);
                subclassTable.setOpaque(false);
                subclassTable.setPreferredSize(new Dimension (150, 120));
                subclassTable.setTableHeader(null);
                subclassTable.setShowGrid(false);

                subclassTable.getColumnModel().getColumn(0).setPreferredWidth(100
        );
                panel.add(subclassTable);
        }

        return panel;
}

private JTable createEmptyTrainSetPane () {
        TrainTableModel model = new TrainTableModel(0);
        JTable table = new JTable (model);
        return table;
}

private JTable createTrainSetPane () {
        TrainTableModel model =
                new TrainTableModel (dataModel.getTotalTrainSamples());

        ArrayList <Object[]> dataArr = new ArrayList <Object[]> ();
        for (int j=0; j<dataModel.getTrainingSamplesIndex().size(); j++) {
                for (int k=0;
                        k<dataModel.getTrainingSamplesIndex().get(j).size(); k++)
                {
                        Object[] arr = new Object[4];
                        int indexPerClass =
                                dataModel.getTrainingSamplesIndex().get(j).get(k);
                        int index =
                                dataModel.getIndicesPerClass().get(j).get(indexPerCl
                        ass);
                        arr[0] = index;
                        arr[1] = dataModel.getClassNames().get(j);
                        dataArr.add(arr);
                }
        }

        for (int j=0; j<dataArr.size(); j++) {
                model.setValueAt(dataArr.get(j)[0], j, 0);
                model.setValueAt(dataArr.get(j)[1], j, 1);
        }

        JTable table = new JTable (model);
        DefaultTableCellRenderer centerRenderer =
                new DefaultTableCellRenderer();
        centerRenderer.setHorizontalAlignment( JLabel.CENTER );
```

```
			table.getColumnModel().getColumn(0).setCellRenderer( centerRenderer );
			table.getColumnModel().getColumn(1).setCellRenderer( centerRenderer );

			return table;
		}
}

@SuppressWarnings("serial")
class TrainTableModel extends AbstractTableModel {

		String[] columnNames = { "Sample ID", "Class" };
		Object[][] data;

		public TrainTableModel(int rowLength) {
			data = new Object[rowLength][columnNames.length];
		}

		@Override
		public int getColumnCount() {
			return columnNames.length;
		}

		@Override
		public int getRowCount() {
			return data.length;
		}

		@Override
		public Object getValueAt(int row, int column) {
			return data[row][column];
		}

		@Override
		public String getColumnName(int column) {
			return columnNames[column];
		}

		public void setValueAt (Object object, int row, int col) {
			data[row][col] = object;
		}

		public void setColValueAt (String str, int col) {
			columnNames[col] = str;
		}
}
```

S. LCCT/src/ui/ResultMarkerGenesTab.java

```
package ui;
```

```
import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.util.ArrayList;

import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableColumn;

import model.SettingsModel;
import data.Dataset;

@SuppressWarnings("serial")
public class ResultMarkerGenesTab extends JPanel {

	private SettingsModel settings;
	private Dataset dataModel;

	public ResultMarkerGenesTab(SettingsModel settings, Dataset dataModel) {
		this.settings = settings;
		this.dataModel = dataModel;

		GridBagLayout gridbag = new GridBagLayout ();
		GridBagConstraints c = new GridBagConstraints ();
		setLayout (gridbag);
		setBackground(Color.WHITE);

		c.weightx = 1;
		c.weighty = 1000;
		c.fill = GridBagConstraints.BOTH;;
		c.gridx = 0;
		c.gridy = 0;
		c.gridwidth = 1;
		c.gridheight = 1;

		JTabbedPane tabpane = new JTabbedPane ();
		tabpane.setBorder (BorderFactory.createEmptyBorder());
		tabpane.setRequestFocusEnabled(false);
		for (int j=0; j<dataModel.getMarkersPerClass().size(); j++) {
			JScrollPane scrollPane = new JScrollPane (createSnrTab (j),
			JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
			JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
			tabpane.addTab(dataModel.getClassNames().get(j), scrollPane);
		}
		gridbag.setConstraints(tabpane, c);
		add (tabpane);
	}

	private JPanel createSnrTab(int i) {
		GridBagLayout gridbag = new GridBagLayout ();
```

```
GridBagConstraints c = new GridBagConstraints ();
JPanel panel = new JPanel ();
panel.setLayout(gridbag);

String[] geneDesc = settings.getGeneDescription (dataModel);


ArrayList <String> classNames = dataModel.getClassNames();

for (int j=0; j<dataModel.getMarkersPerClass().size(); j++) {
       int pos = 0;
       int classIndex = 0;
       for (int w=0; w<dataModel.getMarkersPerClass().get(j).size();
       w++) {

              c.weightx = 0;
              c.weighty = 0;
              c.fill = GridBagConstraints.BOTH;
              if (i==w)
                     classIndex++;
              if (w==dataModel.getMarkersPerClass().size()-1)
                     classIndex-=1;
              JPanel panel1 = new JPanel ();
              panel1.setBackground(Color.decode("#87CEFA"));
              JLabel label = new JLabel (
              "Signal-to-Noise Ratios for class " + classNames.get(i)
              + " and " + classNames.get(classIndex));
              panel1.add(label);
              classIndex++;
              c.gridx = 0;
              c.gridy = pos;
              pos++;

              c.gridwidth = 1;
              c.gridheight = 1;
              gridbag.setConstraints(panel1, c);
              panel.add (panel1);

              c.weightx = 1;
              c.weighty = 1000;
              c.fill = GridBagConstraints.BOTH;
              MarkerGenesTableModel model =
              new MarkerGenesTableModel (settings.getMgCount());

              int index = 0;
              for (int x=0;
              x<dataModel.getMarkersPerClass().get(j).get(w).length;
              x++) {

                     model.setValueAt(dataModel.getMarkersPerClass().get(
                     j).get(w)[x][1], index, 0);
                     model.setValueAt(dataModel.getProbeSet()[(int)
                     dataModel.getMarkersPerClass().get(j).get(w)[x][0]],
                     index, 1);
                     model.setValueAt(geneDesc[(int)
```

```
                                dataModel.getMarkersPerClass().get(j).get(w)[x][0]],
                                index, 2);
                                index++;
                        }

                        JTable table = new JTable (model);
                        TableColumn col0 = table.getColumnModel().getColumn(0);
                        col0.setPreferredWidth(50);
                        TableColumn col1 = table.getColumnModel().getColumn(1);
                        col1.setPreferredWidth(50);
                        TableColumn col2 = table.getColumnModel().getColumn(2);
                        col2.setPreferredWidth(300);

                        c.gridx = 0;
                        c.gridy = pos;
                        pos++;
                        c.gridwidth = 1;
                        c.gridheight = 1;
                        gridbag.setConstraints(table, c);
                        panel.add (table);
                    }
                }

                return panel;
        }
}

@SuppressWarnings("serial")
class MarkerGenesTableModel extends AbstractTableModel {

        String[] columnNames = {"SNR Value", "Probe Set", "Gene Description"};
        Object[][] data;
        int length;

        public MarkerGenesTableModel(int length) {
                this.length = length;
                data = new Object[length][3];
        }

        @Override
        public int getColumnCount() {
                return columnNames.length;
        }

        @Override
        public int getRowCount() {
                return data.length;
        }

        @Override
        public Object getValueAt(int row, int column) {
                return data[row][column];
        }

        @Override
```

```
    public String getColumnName(int column) {
        return columnNames[column];
    }

       public void setValueAt (Object object, int row, int col) {
              data[row][col] = object;
       }
}
```

T. LCCT/src/ui/ResultParametersTab.java

```
package ui;

import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;

import javax.swing.BorderFactory;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableColumn;

import libsvm.svm_parameter;
import model.SettingsModel;

@SuppressWarnings("serial")
public class ResultParametersTab extends JPanel {

       public ResultParametersTab(svm_parameter p, SettingsModel settings) {
              GridBagLayout gridbag = new GridBagLayout ();
              GridBagConstraints c = new GridBagConstraints ();
              setLayout(gridbag);
              setBackground (Color.WHITE);

              c.weightx = 1000;
              c.weighty = 1000;
              c.fill = GridBagConstraints.BOTH;;
              c.gridx = 0;
              c.gridy = 0;
              c.gridwidth = 1;
              c.gridheight = 1;

              ParameterTableModel model = new ParameterTableModel ();

              switch (p.svm_type) {
                     case 0: model.setValueAt("C_SVC", 0, 1); break;
                     case 1: model.setValueAt("NU_SVC", 0, 1); break;
                     case 2: model.setValueAt("ONE CLASS", 0, 1); break;
```

```
        case 3: model.setValueAt("EPSILON_SVR", 0, 1); break;
        case 4: model.setValueAt("NU_SVR", 0, 1); break;
}

switch (p.kernel_type) {
        case 0: model.setValueAt("Linear", 1, 1); break;
        case 1: model.setValueAt("Polynomial", 1, 1); break;
        case 2: model.setValueAt("RBF", 1, 1); break;
        case 3: model.setValueAt("Sigmoid", 1, 1); break;
        case 4: model.setValueAt("Precomputed", 1, 1); break;
}

model.setValueAt(p.probability, 2, 1);
model.setValueAt(p.gamma, 3, 1);
model.setValueAt(p.nr_weight, 4, 1);
model.setValueAt(p.C, 5, 1);
model.setValueAt(p.cache_size, 6, 1);
model.setValueAt(p.eps, 7, 1);
model.setValueAt(p.nr_weight, 8, 1);

String prep = "";
if (settings.getPrep()[0])
        prep += "Decimal Scale Normalization, ";
if (settings.getPrep()[1])
        prep += "Quantile Normalization, ";
if (settings.getPrep()[2])
        prep += "Z-Score Transformation, ";
if (settings.getPrep()[3])
        prep += "Min-Max Normalization, ";
prep = prep.substring(0, prep.lastIndexOf(","));
model.setValueAt(prep, 9, 1);

String gs = "";
if (settings.getGs()[0])
        gs += "Median Absolute Deviation (MAD)";
else
        gs += "Standard Deviation (SD)";
model.setValueAt(gs, 10, 1);

if (settings.getSnr())
        model.setValueAt("Signal-to-Noise Ratio", 11, 1);
else {
        model.setValueAt("None Applied", 11, 1);
}

model.setValueAt(settings.getGsCount(), 12, 1);
model.setValueAt(settings.getMgCount(), 13, 1);

JTable table = new JTable (model);
TableColumn col0 = table.getColumnModel().getColumn(0);
col0.setPreferredWidth(25);
TableColumn col1 = table.getColumnModel().getColumn(1);
col1.setPreferredWidth(250);
JScrollPane scrollPane = new JScrollPane (table,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
```

```
                JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
                scrollPane.setBorder(BorderFactory.createEmptyBorder());

                gridbag.setConstraints(scrollPane, c);
                add (scrollPane);
        }
}

@SuppressWarnings("serial")
class ParameterTableModel extends AbstractTableModel {

        String[] columnNames = {"Parameter", "Value"};
        Object[][] data = {
                        {"SVM Type", null},
                        {"Kernel Type", null},
                        {"Probability", null},
                        {"Gamma", null},
                        {"Nu", null},
                        {"C", null},
                        {"Cache Size", null},
                        {"Stopping Criteria", null},
                        {"NR Weight", null},
                        {"Preprocessing Methods", null},
                        {"Gene Selection Methods", null},
                        {"Marker Gene Identification Method", null},
                        {"No. of Selected Genes", null},
                        {"No. of Identified Marker Genes", null}
        };

        @Override
        public int getColumnCount() {
                return columnNames.length;
        }

        @Override
        public int getRowCount() {
                return data.length;
        }

        @Override
        public Object getValueAt(int row, int column) {
                return data[row][column];
        }

        @Override
        public String getColumnName(int column) {
                return columnNames[column];
        }

        public void setValueAt (Object object, int row, int col) {
                data[row][col] = object;
        }
}
```

U. LCCT/src/ui/ResultPredictionTab.java

```
package ui;

import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.util.ArrayList;

import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableCellRenderer;

import data.Dataset;

@SuppressWarnings("serial")
public class ResultPredictionTab extends JPanel {

	public ResultPredictionTab(Dataset dataModel) {

		GridBagLayout gridbag = new GridBagLayout ();
		GridBagConstraints c = new GridBagConstraints ();
		setLayout(gridbag);
		setBackground (Color.WHITE);

		c.weightx = 1000;
		c.weighty = 1000;
		c.fill = GridBagConstraints.BOTH;;
		c.gridx = 0;
		c.gridy = 0;
		c.gridwidth = 1;
		c.gridheight = 1;

		ArrayList <Object[]> dataArr = new ArrayList <Object[]> ();

		int count = 0;
		for (int j=0; j<dataModel.getTestingSamplesIndex().size(); j++) {
			for (int k=0; k<dataModel.getTestingSamplesIndex().get(j).size();
			k++) {
				Object[] arr = new Object[4];
				int indexPerClass =
				dataModel.getTestingSamplesIndex().get(j).get(k);
				int index =
				dataModel.getIndicesPerClass().get(j).get(indexPerClass);
				int predIndex =
				(int)dataModel.getTrainingResults()[count][1];
				arr[0] = index;
				arr[1] =
				dataModel.getClassNames().
				get((int)dataModel.getTrainingResults()[count][0]);
				arr[2] = dataModel.getClassNames().get(predIndex);
```

```
                        arr[3] =
                        Math.round((dataModel.getTrainingResults()
                        [count][predIndex+2]) * 10000) / (double)100 + "%";
                        dataArr.add(arr);
                        count++;
                  }
            }

            PredictionTableModel trainModel =
                  new PredictionTableModel (dataArr.size());

            for (int j=0; j<dataArr.size(); j++) {
                  trainModel.setValueAt(dataArr.get(j)[0], j, 0);
                  trainModel.setValueAt(dataArr.get(j)[1], j, 1);
                  trainModel.setValueAt(dataArr.get(j)[2], j, 2);
                  trainModel.setValueAt(dataArr.get(j)[3], j, 3);
            }

            JTable table = new JTable (trainModel);
            DefaultTableCellRenderer centerRenderer =
                  new DefaultTableCellRenderer();
            centerRenderer.setHorizontalAlignment( JLabel.CENTER );
            table.getColumnModel().getColumn(0).setCellRenderer( centerRenderer );
            table.getColumnModel().getColumn(1).setCellRenderer( centerRenderer );
            table.getColumnModel().getColumn(2).setCellRenderer( centerRenderer );
            table.getColumnModel().getColumn(3).setCellRenderer( centerRenderer );
            table.setGridColor(Color.DARK_GRAY);

            JScrollPane scrollPane = new JScrollPane (table,
            JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
            JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

            gridbag.setConstraints(scrollPane, c);
            add (scrollPane);

      }
}

@SuppressWarnings("serial")
class PredictionTableModel extends AbstractTableModel {

      String[] columnNames = {"Sample ID", "Actual Class", "Predicted Class",
            "% Probability"};
      Object[][] data;
      int length;

      public PredictionTableModel(int length) {
            this.length = length;
            data = new Object[length][4];
      }

      @Override
      public int getColumnCount() {
            return columnNames.length;
      }
```

```
	@Override
	public int getRowCount() {
		return data.length;
	}

	@Override
	public Object getValueAt(int row, int column) {
		return data[row][column];
	}

	@Override
	public String getColumnName(int column) {
		return columnNames[column];
	}

	public void setValueAt (Object object, int row, int col) {
		data[row][col] = object;
	}
}
```

V. LCCT/src/ui/ResultSelectedGenesTab.java

```
package ui;

import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;

import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableColumn;

import data.Dataset;
import model.SettingsModel;

@SuppressWarnings("serial")
public class ResultSelectedGenesTab extends JPanel {

	public ResultSelectedGenesTab (JFrame appFrame, SettingsModel settings,
		Dataset dataModel) {
		GridBagLayout gridbag = new GridBagLayout ();
		GridBagConstraints c = new GridBagConstraints ();
		setLayout(gridbag);
		setBackground(Color.WHITE);

		c.weightx = 1000;
		c.weighty = 1000;
```

```
			c.fill = GridBagConstraints.BOTH;;
			c.gridx = 0;
			c.gridy = 0;
			c.gridwidth = 1;
			c.gridheight = 1;

			GeneTableModel geneModel = new GeneTableModel (settings.getGsCount());

			String[] geneDescription = settings.getGeneDescription (dataModel);

			for (int j=0; j<settings.getGsCount(); j++) {
				int index = dataModel.getSelectedGenes()[j];
				geneModel.setValueAt(j+1, j, 0);
				geneModel.setValueAt(index, j, 1);
				geneModel.setValueAt(dataModel.getRankedValues()[index], j, 2);
				geneModel.setValueAt(dataModel.getProbeSet()[index], j, 3);
				if (geneDescription.length > 0) {
					geneModel.setValueAt(geneDescription[index], j, 4);
				}
			}

			JTable table = new JTable (geneModel);
			TableColumn col1 = table.getColumnModel().getColumn(4);
			col1.setPreferredWidth(270);
			JScrollPane scrollPane = new JScrollPane (table,
			JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
			JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
			scrollPane.setBorder(BorderFactory.createEmptyBorder());

			gridbag.setConstraints(scrollPane, c);
			add (scrollPane);
		}
}

@SuppressWarnings("serial")
class GeneTableModel extends AbstractTableModel {

		String[] columnNames = {"Rank", "Gene ID", "Value", "Gene Code",
		"Gene Description"};
		Object[][] data;
		int length;

		public GeneTableModel(int length) {
			this.length = length;
			data = new Object[length][5];
		}

		@Override
		public int getColumnCount() {
			return columnNames.length;
		}

		@Override
		public int getRowCount() {
			return data.length;
```

```
	}

	@Override
	public Object getValueAt(int row, int column) {
		return data[row][column];
	}

	@Override
	public String getColumnName(int column) {
		return columnNames[column];
	}

	public void setValueAt (Object object, int row, int col) {
		data[row][col] = object;
	}

}
```

W. LCCT/src/ui/ResultStatsTab.java

```
package ui;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;

import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableCellRenderer;

import data.Dataset;

@SuppressWarnings("serial")
public class ResultStatsTab extends JPanel {

	private Dataset dataModel;
	private int[][] matrix;
	private double accuracy = 0;
	private double[] precision, sensitivity, specificity, negativePredictiveValue;

	public ResultStatsTab(Dataset dataModel) {
```

```
        this.dataModel = dataModel;

        compute ();

        GridBagLayout gridbag = new GridBagLayout ();
        GridBagConstraints c = new GridBagConstraints ();
        setLayout(gridbag);

        c.weightx = 0;
        c.weighty = 0;
        c.fill = GridBagConstraints.BOTH;;
        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;

        JPanel panel = createMatrixPanel ();
        panel.setBackground (Color.WHITE);
        panel.setBorder(BorderFactory.createTitledBorder("Confusion Matrix"));

        gridbag.setConstraints(panel, c);
        add (panel);

        c.weightx = 1000;
        c.weighty = 1000;
        c.fill = GridBagConstraints.BOTH;
        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;

        JPanel detailsPanel = createDetailsPanel ();
        gridbag.setConstraints(detailsPanel, c);
        add (detailsPanel);
}

private JPanel createDetailsPanel () {
        JPanel panel = new JPanel ();
        GridBagLayout gridbag = new GridBagLayout ();
        GridBagConstraints c = new GridBagConstraints ();
        panel.setLayout(gridbag);
        panel.setBorder(BorderFactory.createTitledBorder
                ("Performance Measures"));


        c.weightx = 1;
        c.weighty = 1;
        c.fill = GridBagConstraints.BOTH;;
        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;

        JPanel detailsPanel = new JPanel ();
        detailsPanel.setBackground(Color.white);
```

```
        JLabel accuracyLabel = new JLabel ("Accuracy: ");
        accuracyLabel.setPreferredSize(new Dimension (60, 25));
        JLabel accuracyValue = new JLabel (Double.toString(accuracy) + "%");
        detailsPanel.add(accuracyLabel);
        detailsPanel.add(accuracyValue);

        gridbag.setConstraints(detailsPanel, c);
        panel.add(detailsPanel);

        c.weightx = 1000;
        c.weighty = 1000;
        c.fill = GridBagConstraints.BOTH;;
        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;

        MeasuresTableModel model = new MeasuresTableModel (matrix.length);

        for (int i=0; i<matrix.length; i++) {
              model.setValueAt(dataModel.getClassNames().get(i), i, 0);
              model.setValueAt(precision[i], i, 1);
              model.setValueAt(sensitivity[i], i, 2);
              model.setValueAt(specificity[i], i, 3);
              model.setValueAt(negativePredictiveValue[i], i, 4);
        }
        JTable table = new JTable (model);

        DefaultTableCellRenderer centerRenderer =
        new DefaultTableCellRenderer();
        centerRenderer.setHorizontalAlignment( JLabel.CENTER );
        table.getColumnModel().getColumn(0).setCellRenderer( centerRenderer );
        table.getColumnModel().getColumn(1).setCellRenderer( centerRenderer );

        JScrollPane scrollPane = new JScrollPane (table,
        JScrollPane.VERTICAL_SCROLLBAR_NEVER,
        JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        scrollPane.setPreferredSize(new Dimension (500, 200));
        gridbag.setConstraints(scrollPane, c);
        panel.add(scrollPane);
        return panel;
}

private JPanel createMatrixPanel () {
        JPanel panel = new JPanel ();
        GridBagLayout gridbag = new GridBagLayout ();
        GridBagConstraints c = new GridBagConstraints ();
        panel.setLayout(gridbag);

        c.weightx = 1000;
        c.weighty = 1000;
        c.fill = GridBagConstraints.BOTH;;
        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
```

```
c.gridheight = 1;

int h = 240, w = 620;
if (dataModel.getTotalClasses() <= 5) {
      h = 140;
      w = 400;
}

JLayeredPane lpane = new JLayeredPane ();
lpane.setPreferredSize (new Dimension (w,h));
MatrixPanel matrixPanel = new MatrixPanel (matrix,
dataModel.getClassNames(), w, h,
dataModel.getTotalClasses()+1,dataModel.getTotalClasses()+1);
matrixPanel.setBounds(0, 0, w, h);
lpane.add(matrixPanel);

gridbag.setConstraints(lpane, c);
panel.add(lpane);

c.weightx = 0;
c.weighty = 0;
c.fill = GridBagConstraints.BOTH;
c.gridx = 0;
c.gridy = 1;
c.gridwidth = 1;
c.gridheight = 1;

JPanel legend = new JPanel ();
legend.setBackground(Color.white);
legend.setLayout(new FlowLayout(FlowLayout.LEADING));
JLabel conditionLabel = new JLabel ("Condition / True Class");
JLabel outcomeLabel = new JLabel ("Test Outcome / Predicted Class");
JLabel hitLabel = new JLabel ("Correct Prediction");
JLabel missLabel = new JLabel ("Incorrect Prediction");

JPanel yellowPanel = new JPanel ();
yellowPanel.setBackground(Color.yellow);
yellowPanel.setPreferredSize(new Dimension (20, 15));
yellowPanel.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
JPanel cyanPanel = new JPanel ();
cyanPanel.setBackground(Color.cyan);
cyanPanel.setPreferredSize(new Dimension (20, 15));
cyanPanel.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
JPanel pinkPanel = new JPanel ();
pinkPanel.setBackground(Color.pink);
pinkPanel.setPreferredSize(new Dimension (20, 15));
pinkPanel.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
JPanel whitePanel = new JPanel ();
whitePanel.setBackground(Color.white);
whitePanel.setPreferredSize(new Dimension (20, 15));
whitePanel.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));

legend.add(pinkPanel);
legend.add(outcomeLabel);
legend.add(cyanPanel);
```

```
		legend.add(conditionLabel);
		legend.add(yellowPanel);
		legend.add(hitLabel);
		legend.add(whitePanel);
		legend.add(missLabel);

		gridbag.setConstraints(legend, c);
		panel.add(legend);

		return panel;
	}

	private void compute () {
		double[][] results = dataModel.getTrainingResults();

		matrix = new int[dataModel.getTotalClasses()][];
		for (int i=0; i<dataModel.getTotalClasses(); i++) {
			int counters[] = new int[dataModel.getClassNames().size()];
			for (int j=0; j<counters.length; j++) {
				counters[j] = 0;
			}
			for (int j=0; j<results.length; j++) {
				if (results[j][0] == i) {
					for (int k=0; k<dataModel.getTotalClasses(); k++) {
						if (results[j][1] == k)
							counters[k] += 1;
					}
				}
			}
			matrix[i] = counters;
		}

		int trueValues = 0;
		int total = 0;
		for (int i=0; i<matrix.length; i++) {
			for (int j=0; j<matrix[i].length; j++) {
				if (i==j) {
					trueValues += matrix[i][j];
					total += matrix[i][j];
				} else
					total += matrix[i][j];
			}
		}

		accuracy = Math.round( ((double) trueValues / (double) total) * 10000) /
(double)100;

		precision = new double[matrix.length];
		sensitivity = new double[matrix.length];
		specificity = new double[matrix.length];
		negativePredictiveValue = new double[matrix.length];
		int[] testOutcomPositive = new int[matrix.length];
		int[] conditionPositive = new int[matrix.length];
		int[] truePositive = new int[matrix.length];
		int[] falsePositive = new int[matrix.length];
```

```
int[] trueNegative = new int[matrix.length];
int[] totalPerRow = new int[matrix.length];
int[] totalPerCol = new int[matrix.length];

for (int i=0; i<falsePositive.length; i++) {
	falsePositive[i] = 0;
	trueNegative[i] = 0;
	precision[i] = 0;
	sensitivity[i] = 0;
	specificity[i] = 0;
	negativePredictiveValue[i] = 0;
}

for (int i=0; i<matrix.length; i++) {
	for (int j=0; j<matrix.length; j++) {
		if (i==j) {
			truePositive[i] = matrix[i][j];
		} else {
			falsePositive[i] += matrix[i][j];
		}
	}
}

for (int i=0; i<matrix.length; i++) {
	testOutcomPositive[i] = 0;
	conditionPositive[i] = 0;
	for (int j=0; j<matrix[i].length; j++) {
		conditionPositive[i] += matrix[i][j];
		testOutcomPositive[i] += matrix[j][i];
	}
}

int totalTrueNegative = 0;
for (int i=0; i<truePositive.length; i++) {
	totalTrueNegative += truePositive[i];
}

for (int i=0; i<truePositive.length; i++) {
	trueNegative[i] = totalTrueNegative - truePositive[i];
}

for (int i=0; i<matrix.length; i++) {
	for (int j=0; j<matrix.length; j++) {
		totalPerRow[i] += matrix[i][j];
		totalPerCol[j] += matrix[i][j];
	}
}

int [] conditionNegative = new int[matrix.length];
int [] testOutcomNegative = new int[matrix.length];

int skip = 0;
while (skip != 5) {
	conditionNegative[skip] = 0;
	testOutcomNegative[skip] = 0;
```

```
                for (int i=0; i<matrix.length; i++) {
                        if (skip != i) {
                                for (int j=0; j<matrix.length; j++) {
                                        if (i!=j) {
                                                conditionNegative[skip] +=
                                                matrix[i][j];
                                                testOutcomNegative[skip] +=
                                                matrix[j][i];
                                        }
                                }
                        }
                }
                skip++;
        }


        for (int i=0; i<matrix.length; i++) {
                precision[i] = Math.round(((double)truePositive[i] /
                (double)testOutcomPositive[i])*(double)100) / (double)100;
                sensitivity[i] = Math.round(((double)truePositive[i] /
                (double)conditionPositive[i])*(double)100) / (double)100;
                specificity[i] = Math.round(((double)trueNegative[i] /
                (double)conditionNegative[i])*(double)100) / (double)100;
                negativePredictiveValue[i] = Math.round((double)trueNegative[i] /
                (double)testOutcomNegative[i]) / (double)100;
        }
    }
}

@SuppressWarnings("serial")
class MeasuresTableModel extends AbstractTableModel {

    String[] columnNames = {"Class", "Precision", "Sensitivy (Recall)",
            "Specificity", "Negative Predictive Value"};
    Object[][] data;
    int length;

    public MeasuresTableModel(int length) {
            this.length = length;
            data = new Object[length][5];
    }

    @Override
    public int getColumnCount() {
            return columnNames.length;
    }

    @Override
    public int getRowCount() {
            return data.length;
    }

    @Override
    public Object getValueAt(int row, int column) {
            return data[row][column];
```

```
	}

	@Override
	public String getColumnName(int column) {
		return columnNames[column];
	}

	public void setValueAt (Object object, int row, int col) {
		data[row][col] = object;
	}
}
```

X. LCCT/src/ui/HelpFrame.java

```
package ui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import java.util.ArrayList;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.JTree;
import javax.swing.event.TreeSelectionEvent;
import javax.swing.event.TreeSelectionListener;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreePath;

import model.SettingsModel;
import proc.LcctProcessor;
import util.Utils;
import data.Dataset;
```

```
@SuppressWarnings("serial")
public class SidePanel extends JPanel implements ActionListener, ItemListener,
TreeSelectionListener {
	public int fileIndex = 1;

	private GridBagLayout gridbag = new GridBagLayout ();
	private GridBagConstraints c = new GridBagConstraints ();

	private JFrame appFrame;
	private JButton newBtn, resultListBtn;
	private JButton okBtn, openDatasetBtn, openDescBtn;
	private JTextField fileTextField, descTextField;
	private JLayeredPane newLPane;
	private JLayeredPane resultListPane;
	private JCheckBox classifyCheckBox, subClassifyCheckBox;
	private JCheckBox decScaleCheckBox, qNormCheckBox, zScoreCheckBox,
minMaxNormCheckBox;
	private JCheckBox snrCheckBox, sdCheckBox, madCheckBox;
	private JTextField minField, maxField, noOfMgGenesField, noOfGsGenesField;
	private JProgressBar progressBar;
	private JTabbedPane resultTabbedPane;
	private JTree tree;
	private DefaultMutableTreeNode root;
	private ArrayList <LcctProcessor> processor;
	private JButton startBtn, stopBtn;

	public SidePanel (JFrame appFrame, JTabbedPane resultTabbedPane,
	JProgressBar progressBar) {
		this.appFrame = appFrame;
		this.resultTabbedPane = resultTabbedPane;
		this.progressBar = progressBar;
		this.processor = new ArrayList <LcctProcessor> ();

		setPreferredSize (new Dimension (237,24));
		setOpaque (true);
		setBackground (Color.white);
		setLayout (gridbag);
		createSidePanel ();
	}

	private void createSidePanel () {
		c.weightx = 0;
		c.weighty = 0;
		c.fill = GridBagConstraints.BOTH;
		JPanel labPanel1 = new JPanel (new BorderLayout ());
		labPanel1.setBackground(Color.decode("#B0E0E6"));

		labPanel1.setBorder(BorderFactory.createLineBorder(
		Color.decode("#D0D0D0")));
		JLabel lab1 = new JLabel (" Control Panel");
		lab1.setFont(new Font ("Tahoma", Font.PLAIN, 13));
		labPanel1.add(lab1, BorderLayout.WEST);
```

```
        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        gridbag.setConstraints(labPanel1, c);
        add (labPanel1);

        c.weightx = 1000;
        c.weighty = 1;
        c.fill = GridBagConstraints.BOTH;

        JLayeredPane controlPanel = new JLayeredPane ();
        newLPane = createNewFilePane ();
        newLPane.setBounds(0, 0, 235, 500);
        resultListPane = createResultsPane ();
        resultListPane.setBounds(0, 0, 235, 500);
        resultListPane.setVisible(false);
        controlPanel.add(newLPane);
        controlPanel.add(resultListPane);


        newBtn = new JButton ("Input New Dataset");
        newBtn.addActionListener(this);
        newBtn.setBounds(0, 0, 238, 25);
        newBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
        newBtn.setBackground(Color.CYAN);
        resultListBtn = new JButton ("Result List");
        resultListBtn.addActionListener(this);

        resultListBtn.setBorder(BorderFactory.createLineBorder(
        Color.LIGHT_GRAY));
        resultListBtn.setBackground(Color.LIGHT_GRAY);

        controlPanel.add(newBtn);
        controlPanel.add(resultListBtn);

        controlPanel.setBorder(BorderFactory.createEmptyBorder());
        controlPanel.setRequestFocusEnabled(false);
        controlPanel.setBackground(Color.decode("#87CEFA"));

        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 1;
        gridbag.setConstraints(controlPanel, c);
        add (controlPanel);
}

private JLayeredPane createResultsPane () {
        JLayeredPane resultListPane = new JLayeredPane ();
        resultListPane.setBorder(BorderFactory.createLineBorder(Color.gray));

        root = new DefaultMutableTreeNode ("Imported Datasets");
        tree = new JTree(root);
        tree.setRootVisible(true);
```

```
        tree.setBorder(BorderFactory.createLineBorder(Color.LightGray));
        tree.addTreeSelectionListener(this);

        JScrollPane pane = new JScrollPane(tree);
        pane.setBorder(BorderFactory.createEmptyBorder());
        pane.getVerticalScrollBar();
        pane.setBounds(5, 30, 227, 430);
        resultListPane.add(pane);

        startBtn = new JButton ("Start");
        startBtn.setBounds(13, 465, 100, 20);
        startBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
        startBtn.addActionListener(this);
        resultListPane.add(startBtn);
        stopBtn = new JButton ("Stop");
        stopBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
        stopBtn.setBounds(118, 465, 100, 20);
        stopBtn.addActionListener(this);
        resultListPane.add(stopBtn);

        return resultListPane;
    }

    private JLayeredPane createNewFilePane () {
        JLayeredPane newLPane = new JLayeredPane ();
        newLPane.setPreferredSize(new Dimension (237, 500));
        newLPane.setVisible(true);

        openDatasetBtn = new JButton ("Data");

        openDatasetBtn.setBorder(BorderFactory.createLineBorder(
        Color.LIGHT_GRAY));
        openDatasetBtn.setBounds(5, 30, 60, 25);
        openDatasetBtn.addActionListener(this);
        fileTextField = new JTextField ();
        fileTextField.setBounds(70, 30, 160, 25);

        openDescBtn = new JButton ("Desc");
        openDescBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
        openDescBtn.setBounds(5, 60, 60, 25);
        openDescBtn.addActionListener(this);
        descTextField = new JTextField ();
        descTextField.setBounds(70, 60, 160, 25);

        classifyCheckBox = new JCheckBox ("Classify");
        classifyCheckBox.setBounds(30, 100, 100, 25);
        classifyCheckBox.setOpaque(false);
        subClassifyCheckBox = new JCheckBox ("Subclassify");
        subClassifyCheckBox.setBounds(130, 100, 80, 25);
        subClassifyCheckBox.setOpaque(false);

        JLabel prepLabel = new JLabel ("Preprocessing Method: ");
        prepLabel.setBounds(10, 130, 200, 25);
        decScaleCheckBox = new JCheckBox ("Decimal Scale Normalization");
        decScaleCheckBox.setSelected(true);
```

```
decScaleCheckBox.setBounds(20, 150, 200, 25);
decScaleCheckBox.setOpaque(false);
qNormCheckBox = new JCheckBox ("Quantile Normalization");
qNormCheckBox.setBounds(20, 170, 200, 25);
qNormCheckBox.setOpaque(false);
zScoreCheckBox = new JCheckBox ("Z-Score Transformation");
zScoreCheckBox.setBounds(20, 190, 200, 25);
zScoreCheckBox.setOpaque(false);
minMaxNormCheckBox = new JCheckBox ("Min-Max Normalization");
minMaxNormCheckBox.setBounds(20, 210, 200, 25);
minMaxNormCheckBox.addItemListener(this);
minMaxNormCheckBox.setOpaque(false);
JLabel minLabel = new JLabel ("Min:");
minLabel.setBounds(40, 235, 50, 25);
minField = new JTextField ("0");
minField.setBounds(70, 235, 50, 20);
minField.setEnabled(false);
JLabel maxLabel = new JLabel ("Max:");
maxLabel.setBounds(40, 260, 50, 25);
maxField = new JTextField ("100");
maxField.setBounds(70, 260, 50, 20);
maxField.setEnabled(false);

JLabel gsLabel = new JLabel ("Gene Selection Method: ");
gsLabel.setBounds(10, 290, 200, 25);
JLabel noOfGsGenesLabel = new JLabel ("# of Genes: ");
noOfGsGenesLabel.setBounds(30, 310, 200, 25);
noOfGsGenesField = new JTextField ("1000");
noOfGsGenesField.setBounds(100, 310, 50, 20);
madCheckBox = new JCheckBox ("Median Absolute Deviation");
madCheckBox.setSelected(true);
madCheckBox.setBounds(20, 330, 200, 25);
madCheckBox.setOpaque(false);
sdCheckBox = new JCheckBox ("Standard Deviation");
sdCheckBox.setBounds(20, 350, 200, 25);
sdCheckBox.setOpaque(false);

JLabel mgLabel = new JLabel ("Marker Gene Selection: ");
mgLabel.setBounds(10, 380, 200, 25);
JLabel noOfMgGenesLabel = new JLabel ("# of Genes: ");
noOfMgGenesLabel.setBounds(30, 400, 200, 25);
noOfMgGenesField = new JTextField ("50");
noOfMgGenesField.setBounds(100, 400, 50, 20);
snrCheckBox = new JCheckBox ("Signal-to-Noise Ratio");
snrCheckBox.setSelected(true);
snrCheckBox.setBounds(20, 420, 200, 25);
snrCheckBox.setOpaque(false);
snrCheckBox.addItemListener(this);

JButton resetBtn = new JButton ("Reset");
resetBtn.setBounds(15, 455, 100, 25);
resetBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
okBtn = new JButton ("OK");
okBtn.addActionListener(this);
okBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
```

```
        okBtn.setBounds(125, 455, 100, 25);

        newLPane.add(openDatasetBtn);
        newLPane.add(fileTextField);
        newLPane.add(openDescBtn);
        newLPane.add(descTextField);
        newLPane.add(classifyCheckBox);
        newLPane.add(subClassifyCheckBox);
        newLPane.add(prepLabel);
        newLPane.add(decScaleCheckBox);
        newLPane.add(qNormCheckBox);
        newLPane.add(zScoreCheckBox);
        newLPane.add(minMaxNormCheckBox);
        newLPane.add(minLabel);
        newLPane.add(minField);
        newLPane.add(maxLabel);
        newLPane.add(maxField);
        newLPane.add(gsLabel);
        newLPane.add(noOfGsGenesLabel);
        newLPane.add(noOfGsGenesField);
        newLPane.add(madCheckBox);
        newLPane.add(sdCheckBox);
        newLPane.add(mgLabel);
        newLPane.add(noOfMgGenesLabel);
        newLPane.add(noOfMgGenesField);
        newLPane.add(snrCheckBox);
        newLPane.add(resetBtn);
        newLPane.add(okBtn);

        return newLPane;
    }

    private void addToFilesPane (String dataFile, String descFile) {
        DefaultMutableTreeNode setNode = new DefaultMutableTreeNode
        ("Dataset " +  fileIndex);
        DefaultMutableTreeNode dataNode = new DefaultMutableTreeNode (dataFile);
        DefaultMutableTreeNode descNode = new DefaultMutableTreeNode (descFile);
        setNode.add(dataNode);
        setNode.add(descNode);
        root.add(setNode);
        tree.repaint();
        tree.revalidate();
    }

    @Override
    public void actionPerformed(ActionEvent event) {
        Object src = event.getSource();

        if (src == newBtn) {
            newBtn.setBackground(Color.cyan);
            resultListBtn.setBackground(Color.LIGHT_GRAY);
            newLPane.setVisible(true);
            resultListPane.setVisible(false);
        }
        if (src == resultListBtn) {
```

```
        resultListBtn.setBackground(Color.cyan);
        newBtn.setBackground(Color.LIGHT_GRAY);
        newLPane.setVisible(false);
        resultListPane.setVisible(true);
}
if (src == openDatasetBtn) {
        Utils.openFileChooser(fileTextField);
}
if (src== openDescBtn) {
        Utils.openFileChooser(descTextField);
}
if (src == okBtn) {
        if (fileTextField.getText().equals("")) {
                JOptionPane.showMessageDialog(appFrame,
                        "Choose an input dataset!",
                        "Warning",
                        JOptionPane.WARNING_MESSAGE);
        }
        else if ((!subClassifyCheckBox.isSelected() &&
                !classifyCheckBox.isSelected())
                || (subClassifyCheckBox.isSelected() &&
                classifyCheckBox.isSelected())) {
                        JOptionPane.showMessageDialog(appFrame,
                        "Select whether to classify or subclassify.",
                            "Warning",
                            JOptionPane.WARNING_MESSAGE);
        }
        else {
                Dataset dataModel = new Dataset (fileTextField.getText(),
                subClassifyCheckBox.isSelected(), appFrame);
                if (!dataModel.hasError()) {
                        addToFilesPane (fileTextField.getText(),
                        descTextField.getText());
                        fileIndex++;
                        SettingsModel settings = new SettingsModel ();
                        settings.setPrep(decScaleCheckBox.isSelected(),
                                qNormCheckBox.isSelected(),
                                zScoreCheckBox.isSelected(),
                                minMaxNormCheckBox.isSelected());
                        settings.setGs(madCheckBox.isSelected(),
                        sdCheckBox.isSelected());
                        if (minMaxNormCheckBox.isSelected())

                        settings.setMinMaxInterval(Integer.parseInt(
                        minField.getText()),
                        Integer.parseInt(maxField.getText()));

                        settings.setGsCount(Integer.parseInt(
                        noOfGsGenesField.getText()));
                        settings.setSnr(snrCheckBox.isSelected());
                        settings.setGeneFile(descTextField.getText());
                        if (snrCheckBox.isSelected())

                        settings.setMgCount(Integer.parseInt(
                        noOfMgGenesField.getText()));
```

```
                        LcctProcessor processor =
                                new LcctProcessor (dataModel,
                        settings, resultTabbedPane, appFrame);
                        processor.addPropertyChangeListener(
                                new PropertyChangeListener () {
                                        public void
                                        propertyChange(PropertyChangeEvent e) {
                                                if
                                                (e.getPropertyName().
                                                equals("progress")) {
                                                progressBar.setValue(
                                                (Integer) e.getNewValue());
                                                }

                                        }

                                });
                        processor.execute();
                }
        }
    }
}

@Override
public void itemStateChanged(ItemEvent e) {
    if (e.getStateChange() == ItemEvent.SELECTED) {
        if (e.getSource() == minMaxNormCheckBox) {
            minField.setEnabled(true);
            maxField.setEnabled(true);
        }
        if (e.getSource() == snrCheckBox)
            noOfMgGenesField.setEnabled(true);
    }
    if (e.getStateChange() == ItemEvent.DESELECTED) {
        if (e.getSource() == minMaxNormCheckBox) {
            minField.setEnabled(false);
            maxField.setEnabled(false);
        }
        if (e.getSource() == snrCheckBox)
            noOfMgGenesField.setEnabled(false);
    }

}

@Override
public void valueChanged(TreeSelectionEvent event) {
    TreePath path = event.getPath();
    int pathCount = path.getPathCount();
    /*String str = path.getLastPathComponent().toString();
    if (str.toString().contains("Dataset ")) {
        System.out.println(str.substring(str.length()-1));
    }*/
```

```
	}
}
```

Y. LCCT/src/ui/UserInterface.java

```
package ui;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JProgressBar;
import javax.swing.JTabbedPane;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;

public class UserInterface implements ActionListener {

	public final int WIDTH = 900;
	public final int HEIGHT = 570;

	private JFrame appFrame;
	private UIManager.LookAndFeelInfo[] looks;
	private JPanel mainPanel;
	private JProgressBar progressBar;
	private JTabbedPane resultTabbedPane;
	private JMenuItem helpMenuItem;

	private GridBagLayout gridbag;
	private GridBagConstraints c;

	public UserInterface () {
		appFrame = new JFrame ("Lung Cancer Classification System");
		appFrame.setSize(WIDTH, HEIGHT);

		looks = UIManager.getInstalledLookAndFeels();
		changeLookAndFeel (3);

		createMenuBar ();
```

```
        resultTabbedPane = new JTabbedPane ();
        resultTabbedPane.setBackground(Color.WHITE);
        resultTabbedPane.addTab ("Dataset", new ResultDatasetTab ());

        for (int i=1; i<resultTabbedPane.getTabCount(); i++) {
              resultTabbedPane.setEnabledAt(i, false);
        }

        mainPanel = new JPanel ();
        gridbag = new GridBagLayout ();
        c = new GridBagConstraints ();

        mainPanel.setLayout(gridbag);
        progressBar = new JProgressBar ();

        c.weightx = 1000;
        c.weighty = 1;
        c.fill = GridBagConstraints.BOTH;
        JPanel mainArea = createMainArea ();
        addComponent (mainArea, 0,0,1,1);

        c.weightx = 0;
        c.weighty = 0;
        c.fill = GridBagConstraints.BOTH;
        progressBar.setValue(0);
        progressBar.setPreferredSize(new Dimension (400, 10));
        addComponent (progressBar, 0,1,1,1);

        appFrame.getContentPane().add(mainPanel, BorderLayout.CENTER);
        appFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        appFrame.setResizable(false);
        appFrame.setVisible(true);
}

private JPanel createMainArea () {
        JPanel mainAreaPanel = new JPanel ();

        GridBagLayout gridbag = new GridBagLayout ();
        GridBagConstraints c = new GridBagConstraints ();

        mainAreaPanel.setLayout (gridbag);

        c.weightx = 0;
        c.weighty = 0;
        c.fill = GridBagConstraints.BOTH;
        JPanel sidebar = new JPanel (new BorderLayout ());
        sidebar.add(new SidePanel (appFrame, resultTabbedPane, progressBar),
        BorderLayout.WEST);
        sidebar.setBackground(Color.white);

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        gridbag.setConstraints(sidebar, c);
```

```
        mainAreaPanel.add (sidebar);

        c.weightx = 1000;
        c.weighty = 1;
        c.fill = GridBagConstraints.BOTH;

        c.gridx = 1;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        JPanel resultPane = new JPanel ();
        resultPane.setBorder(BorderFactory.createLineBorder(Color.gray));
        resultPane.setBackground(Color.LIGHT_GRAY);;
        resultPane.add(resultTabbedPane);

        gridbag.setConstraints(resultPane, c);
        resultTabbedPane.setPreferredSize(new Dimension (639, 496));
        resultTabbedPane.setBorder(BorderFactory.createEmptyBorder());
        resultTabbedPane.setBackground(Color.decode("#87CEFA"));
        resultTabbedPane.setForeground(Color.DARK_GRAY);
        mainAreaPanel.add (resultPane);

        return mainAreaPanel;
}

private void addComponent (Component component, int row, int col, int width,
int height) {
        c.gridx = row;
        c.gridy = col;
        c.gridwidth = width;
        c.gridheight = height;
        gridbag.setConstraints(component, c);
        mainPanel.add (component);
}

private void createMenuBar () {
        JMenu helpMenu = new JMenu ("Help");
        helpMenu.setMnemonic('H');
        helpMenuItem = new JMenuItem ("User Guide");
        helpMenuItem.addActionListener(this);
        helpMenu.add(helpMenuItem);
        JMenuBar menuBar = new JMenuBar ();
        menuBar.add(helpMenu);
        appFrame.setJMenuBar(menuBar);
}

private void changeLookAndFeel (int value) {
        try {
                UIManager.setLookAndFeel(looks[value].getClassName());
                SwingUtilities.updateComponentTreeUI(appFrame);
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

```
	@Override
	public void actionPerformed(ActionEvent e) {
		if (e.getSource() == helpMenuItem) {
			new HelpFrame ();
		}

	}
}
```

Z. LCCT/src/util/Utils.java

```
package util;


import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

import javax.swing.JFileChooser;
import javax.swing.JTextField;
import javax.swing.filechooser.FileNameExtensionFilter;

public class Utils {

	private static BufferedReader br;
	private static String emptyString = "Field is empty.";
	private static String notCsvFile =
	"File is in wrong format. File should be in .CSV.";
	private static String fileNotFound = "File not found.";
	private static String ioException = "Error opening file.";
	private static String valid = "Valid.";

	public static void openFileChooser (JTextField fileTextField) {
		JFileChooser fc1 = new JFileChooser ();
		FileNameExtensionFilter filter = new FileNameExtensionFilter (
		"CSV files", "csv");
		fc1.setFileFilter(filter);
		int result = fc1.showOpenDialog(null);

		if (result == JFileChooser.APPROVE_OPTION) {
			String filePath = null;
			try {
				filePath = fc1.getSelectedFile().toString();
			}
			catch (Exception ex) {
				System.out.println("Could not open file");
			}
			if (filePath != null) {
				fileTextField.setText(filePath);
			}
```

```
		}
	}

	public static String isValidFile (String file) {
		try {
			if ((file.length() > 0 && !file.equals(""))) {
				if (!file.endsWith(".csv")) {
					return notCsvFile;
				}

				br = new BufferedReader (new FileReader(file));
				br.readLine();
			} else return emptyString;
		} catch (FileNotFoundException e) {
			return fileNotFound;
		} catch (IOException e) {
			return ioException;
		} finally {
			try {
				if (br != null)
					br.close();
			} catch (IOException e) {
				return ioException;
			}
		}
		return valid;
	}

	public static String[] convertToStringArray (ArrayList <String> arraylist) {
		String[] stringArr = new String [arraylist.size()];
		for (int i=0; i<arraylist.size(); i++) {
			stringArr[i] = arraylist.get(i);
		}
		return stringArr;
	}
}
```

LIBSVM Source Code**

A. LCCT/libsvm/libsvm/svm_model.java**

```
//
// svm_model
//
package libsvm;
public class svm_model implements java.io.Serializable
{
	public svm_parameter param;	// parameter
	public int nr_class;		// number of classes, = 2 in regression/one
class svm
	public int l;			// total #SV
	public svm_node[][] SV;		// SVs (SV[l])
	public double[][] sv_coef;// coefficients for SVs in decision functions
(sv_coef[k-1][l])
```

```
	public double[] rho;			// constants in decision functions (rho[k*(k-
1)/2])
	public double[] probA;		// pariwise probability information
	public double[] probB;
	public int[] sv_indices;		// sv_indices[0,...,nSV-1] are values in
[1,...,num_traning_data] to indicate SVs in the training set

	// for classification only

	public int[] label;		// label of each class (label[k])
	public int[] nSV;		// number of SVs for each class (nSV[k])
					// nSV[0] + nSV[1] + ... + nSV[k-1] = l
};
```

B. LCCT/libsvm/libsvm/svm_node.java**

```
package libsvm;
public class svm_node implements java.io.Serializable
{
	public int index;
	public double value;
}
```

C. LCCT/libsvm/libsvm/svm_parameter.java**

```
package libsvm;
public class svm_parameter implements Cloneable,java.io.Serializable
{
	/* svm_type */
	public static final int C_SVC = 0;
	public static final int NU_SVC = 1;
	public static final int ONE_CLASS = 2;
	public static final int EPSILON_SVR = 3;
	public static final int NU_SVR = 4;

	/* kernel_type */
	public static final int LINEAR = 0;
	public static final int POLY = 1;
	public static final int RBF = 2;
	public static final int SIGMOID = 3;
	public static final int PRECOMPUTED = 4;

	public int svm_type;
	public int kernel_type;
	public int degree;	// for poly
	public double gamma;	// for poly/rbf/sigmoid
	public double coef0;	// for poly/sigmoid

	// these are for training only
	public double cache_size; // in MB
	public double eps;	// stopping criteria
	public double C;	// for C_SVC, EPSILON_SVR and NU_SVR
```

```
	public int nr_weight;		// for C_SVC
	public int[] weight_label;// for C_SVC
	public double[] weight;		// for C_SVC
	public double nu;	// for NU_SVC, ONE_CLASS, and NU_SVR
	public double p;	// for EPSILON_SVR
	public int shrinking;	// use the shrinking heuristics
	public int probability; // do probability estimates

	public Object clone()
	{
		try
		{
			return super.clone();
		} catch (CloneNotSupportedException e)
		{
			return null;
		}
	}

}
```

D. LCCT/libsvm/libsvm/svm_print_interface.java**

```
package libsvm;
public interface svm_print_interface
{
	public void print(String s);
}
```

E. LCCT/libsvm/libsvm/svm_problem.java**

```
package libsvm;
public class svm_problem implements java.io.Serializable
{
	public int l;
	public double[] y;
	public svm_node[][] x;
}
```

F. LCCT/libsvm/libsvm/svm.java**

```
package libsvm;
import java.io.*;
import java.util.*;

//
// Kernel Cache
//
// l is the number of total data items
// size is the cache size limit in bytes
//
```

```
class Cache {
	private final int l;
	private long size;
	private final class head_t
	{
		head_t prev, next;  // a cicular list
		float[] data;
		int len;            // data[0,len) is cached in this entry
	}
	private final head_t[] head;
	private head_t lru_head;

	Cache(int l_, long size_)
	{
		l = l_;
		size = size_;
		head = new head_t[l];
		for(int i=0;i<l;i++) head[i] = new head_t();
		size /= 4;
		size -= l * (16/4); // sizeof(head_t) == 16
		size = Math.max(size, 2* (long) l);
		// cache must be large enough for two columns
		lru_head = new head_t();
		lru_head.next = lru_head.prev = lru_head;
	}

	private void lru_delete(head_t h)
	{
		// delete from current location
		h.prev.next = h.next;
		h.next.prev = h.prev;
	}

	private void lru_insert(head_t h)
	{
		// insert to last position
		h.next = lru_head;
		h.prev = lru_head.prev;
		h.prev.next = h;
		h.next.prev = h;
	}

	// request data [0,len)
	// return some position p where [p,len) need to be filled
	// (p >= len if nothing needs to be filled)
	// java: simulate pointer using single-element array
	int get_data(int index, float[][] data, int len)
	{
		head_t h = head[index];
		if(h.len > 0) lru_delete(h);
		int more = len - h.len;

		if(more > 0)
		{
			// free old space
```

```
                while(size < more)
                {
                        head_t old = lru_head.next;
                        lru_delete(old);
                        size += old.len;
                        old.data = null;
                        old.len = 0;
                }

                // allocate new space
                float[] new_data = new float[len];
                if(h.data != null) System.arraycopy(h.data,0,new_data,0,h.len);
                h.data = new_data;
                size -= more;
                do {int _=h.len; h.len=len; len=_;} while(false);
        }

        lru_insert(h);
        data[0] = h.data;
        return len;
}

void swap_index(int i, int j)
{
        if(i==j) return;

        if(head[i].len > 0) lru_delete(head[i]);
        if(head[j].len > 0) lru_delete(head[j]);
        do {float[] _=head[i].data; head[i].data=head[j].data; head[j].data=_;
        } while(false);
        do {int _=head[i].len; head[i].len=head[j].len; head[j].len=_;
        } while(false);
        if(head[i].len > 0) lru_insert(head[i]);
        if(head[j].len > 0) lru_insert(head[j]);

        if(i>j) do {int _=i; i=j; j=_;} while(false);
        for(head_t h = lru_head.next; h!=lru_head; h=h.next)
        {
                if(h.len > i)
                {
                        if(h.len > j)
                                do {
                                float _=h.data[i]; h.data[i]=h.data[j]; h.data[j]=_;
                                } while(false);
                        else
                        {
                                // give up
                                lru_delete(h);
                                size += h.len;
                                h.data = null;
                                h.len = 0;
                        }
                }
        }
}
```

```
}

//
// Kernel evaluation
//
// the static method k_function is for doing single kernel evaluation
// the constructor of Kernel prepares to calculate the l*l kernel matrix
// the member function get_Q is for getting one column from the Q Matrix
//
abstract class QMatrix {
	abstract float[] get_Q(int column, int len);
	abstract double[] get_QD();
	abstract void swap_index(int i, int j);
};

abstract class Kernel extends QMatrix {
	private svm_node[][] x;
	private final double[] x_square;

	// svm_parameter
	private final int kernel_type;
	private final int degree;
	private final double gamma;
	private final double coef0;

	abstract float[] get_Q(int column, int len);
	abstract double[] get_QD();

	void swap_index(int i, int j)
	{
		do {svm_node[] _=x[i]; x[i]=x[j]; x[j]=_;} while(false);
		if(x_square != null) do {double _=x_square[i]; x_square[i]=x_square[j];
			x_square[j]=_;} while(false);
	}

	private static double powi(double base, int times)
	{
		double tmp = base, ret = 1.0;

		for(int t=times; t>0; t/=2)
		{
			if(t%2==1) ret*=tmp;
			tmp = tmp * tmp;
		}
		return ret;
	}

	double kernel_function(int i, int j)
	{
		switch(kernel_type)
		{
			case svm_parameter.LINEAR:
				return dot(x[i],x[j]);
			case svm_parameter.POLY:
				return powi(gamma*dot(x[i],x[j])+coef0,degree);
```

```
                case svm_parameter.RBF:
                        return Math.exp(-gamma*(x_square[i]+x_square[j]-
                                2*dot(x[i],x[j])));
                case svm_parameter.SIGMOID:
                        return Math.tanh(gamma*dot(x[i],x[j])+coef0);
                case svm_parameter.PRECOMPUTED:
                        return x[i][(int)(x[j][0].value)].value;
                default:
                        return 0;       // java
        }
}

Kernel(int l, svm_node[][] x_, svm_parameter param)
{
        this.kernel_type = param.kernel_type;
        this.degree = param.degree;
        this.gamma = param.gamma;
        this.coef0 = param.coef0;

        x = (svm_node[][])x_.clone();

        if(kernel_type == svm_parameter.RBF)
        {
                x_square = new double[l];
                for(int i=0;i<l;i++)
                        x_square[i] = dot(x[i],x[i]);
        }
        else x_square = null;
}

static double dot(svm_node[] x, svm_node[] y)
{
        double sum = 0;
        int xlen = x.length;
        int ylen = y.length;
        int i = 0;
        int j = 0;
        while(i < xlen && j < ylen)
        {
                if(x[i].index == y[j].index)
                        sum += x[i++].value * y[j++].value;
                else
                {
                        if(x[i].index > y[j].index)
                                ++j;
                        else
                                ++i;
                }
        }
        return sum;
}

static double k_function(svm_node[] x, svm_node[] y,
                                svm_parameter param)
{
```

```
		switch(param.kernel_type)
		{
			case svm_parameter.LINEAR:
				return dot(x,y);
			case svm_parameter.POLY:
				return
				powi(param.gamma*dot(x,y)+param.coef0,param.degree);
			case svm_parameter.RBF:
			{
				double sum = 0;
				int xlen = x.length;
				int ylen = y.length;
				int i = 0;
				int j = 0;
				while(i < xlen && j < ylen)
				{
					if(x[i].index == y[j].index)
					{
						double d = x[i++].value - y[j++].value;
						sum += d*d;
					}
					else if(x[i].index > y[j].index)
					{
						sum += y[j].value * y[j].value;
						++j;
					}
					else
					{
						sum += x[i].value * x[i].value;
						++i;
					}
				}

				while(i < xlen)
				{
					sum += x[i].value * x[i].value;
					++i;
				}

				while(j < ylen)
				{
					sum += y[j].value * y[j].value;
					++j;
				}

				return Math.exp(-param.gamma*sum);
			}
			case svm_parameter.SIGMOID:
				return Math.tanh(param.gamma*dot(x,y)+param.coef0);
			case svm_parameter.PRECOMPUTED:
				return x[(int)(y[0].value)].value;
			default:
				return 0;	// java
		}
	}
```

```
}

// An SMO algorithm in Fan et al., JMLR 6(2005), p. 1889--1918
// Solves:
//
//     min 0.5(\alpha^T Q \alpha) + p^T \alpha
//
//           y^T \alpha = \delta
//           y_i = +1 or -1
//           0 <= alpha_i <= Cp for y_i = 1
//           0 <= alpha_i <= Cn for y_i = -1
//
// Given:
//
//     Q, p, y, Cp, Cn, and an initial feasible point \alpha
//     l is the size of vectors and matrices
//     eps is the stopping tolerance
//
// solution will be put in \alpha, objective value will be put in obj
//
class Solver {
       int active_size;
       byte[] y;
       double[] G;         // gradient of objective function
       static final byte LOWER_BOUND = 0;
       static final byte UPPER_BOUND = 1;
       static final byte FREE = 2;
       byte[] alpha_status;      // LOWER_BOUND, UPPER_BOUND, FREE
       double[] alpha;
       QMatrix Q;
       double[] QD;
       double eps;
       double Cp,Cn;
       double[] p;
       int[] active_set;
       double[] G_bar;           // gradient, if we treat free variables as 0
       int l;
       boolean unshrink;   // XXX

       static final double INF = java.lang.Double.POSITIVE_INFINITY;

       double get_C(int i)
       {
              return (y[i] > 0)? Cp : Cn;
       }
       void update_alpha_status(int i)
       {
              if(alpha[i] >= get_C(i))
                     alpha_status[i] = UPPER_BOUND;
              else if(alpha[i] <= 0)
                     alpha_status[i] = LOWER_BOUND;
              else alpha_status[i] = FREE;
       }
       boolean is_upper_bound(int i) { return alpha_status[i] == UPPER_BOUND; }
       boolean is_lower_bound(int i) { return alpha_status[i] == LOWER_BOUND; }
```

```
boolean is_free(int i) {  return alpha_status[i] == FREE; }

// java: information about solution except alpha,
// because we cannot return multiple values otherwise...
static class SolutionInfo {
       double obj;
       double rho;
       double upper_bound_p;
       double upper_bound_n;
       double r;    // for Solver_NU
}

void swap_index(int i, int j)
{
       Q.swap_index(i,j);
       do {byte _=y[i]; y[i]=y[j]; y[j]=_;} while(false);
       do {double _=G[i]; G[i]=G[j]; G[j]=_;} while(false);
       do {byte _=alpha_status[i]; alpha_status[i]=alpha_status[j];
        alpha_status[j]=_;} while(false);
       do {double _=alpha[i]; alpha[i]=alpha[j]; alpha[j]=_;} while(false);
       do {double _=p[i]; p[i]=p[j]; p[j]=_;} while(false);
       do {int _=active_set[i]; active_set[i]=active_set[j]; active_set[j]=_;
       } while(false);
       do {double _=G_bar[i]; G_bar[i]=G_bar[j]; G_bar[j]=_;} while(false);
}

void reconstruct_gradient()
{
       // reconstruct inactive elements of G from G_bar and free variables

       if(active_size == l) return;

       int i,j;
       int nr_free = 0;

       for(j=active_size;j<l;j++)
              G[j] = G_bar[j] + p[j];

       for(j=0;j<active_size;j++)
              if(is_free(j))
                     nr_free++;

       if(2*nr_free < active_size)
              svm.info("\nWARNING: using -h 0 may be faster\n");

       if (nr_free*l > 2*active_size*(l-active_size))
       {
              for(i=active_size;i<l;i++)
              {
                     float[] Q_i = Q.get_Q(i,active_size);
                     for(j=0;j<active_size;j++)
                            if(is_free(j))
                                   G[i] += alpha[j] * Q_i[j];
              }
       }
```

```
		else
		{
			for(i=0;i<active_size;i++)
				if(is_free(i))
				{
					float[] Q_i = Q.get_Q(i,l);
					double alpha_i = alpha[i];
					for(j=active_size;j<l;j++)
						G[j] += alpha_i * Q_i[j];
				}
		}
	}

	void Solve(int l, QMatrix Q, double[] p_, byte[] y_,
		double[] alpha_, double Cp, double Cn, double eps, SolutionInfo si,
		int shrinking)
	{
		this.l = l;
		this.Q = Q;
		QD = Q.get_QD();
		p = (double[])p_.clone();
		y = (byte[])y_.clone();
		alpha = (double[])alpha_.clone();
		this.Cp = Cp;
		this.Cn = Cn;
		this.eps = eps;
		this.unshrink = false;

		// initialize alpha_status
		{
			alpha_status = new byte[l];
			for(int i=0;i<l;i++)
				update_alpha_status(i);
		}

		// initialize active set (for shrinking)
		{
			active_set = new int[l];
			for(int i=0;i<l;i++)
				active_set[i] = i;
			active_size = l;
		}

		// initialize gradient
		{
			G = new double[l];
			G_bar = new double[l];
			int i;
			for(i=0;i<l;i++)
			{
				G[i] = p[i];
				G_bar[i] = 0;
			}
			for(i=0;i<l;i++)
				if(!is_lower_bound(i))
```

```
                {
                        float[] Q_i = Q.get_Q(i,l);
                        double alpha_i = alpha[i];
                        int j;
                        for(j=0;j<l;j++)
                                G[j] += alpha_i*Q_i[j];
                        if(is_upper_bound(i))
                                for(j=0;j<l;j++)
                                        G_bar[j] += get_C(i) * Q_i[j];
                }
}

// optimization step

int iter = 0;
int max_iter = Math.max(10000000, l>Integer.MAX_VALUE/100 ?
Integer.MAX_VALUE : 100*l);
int counter = Math.min(l,1000)+1;
int[] working_set = new int[2];

while(iter < max_iter)
{
        // show progress and do shrinking

        if(--counter == 0)
        {
                counter = Math.min(l,1000);
                if(shrinking!=0) do_shrinking();
                svm.info(".");
        }

        if(select_working_set(working_set)!=0)
        {
                // reconstruct the whole gradient
                reconstruct_gradient();
                // reset active set size and check
                active_size = l;
                svm.info("*");
                if(select_working_set(working_set)!=0)
                        break;
                else
                        counter = 1; // do shrinking next iteration
        }

        int i = working_set[0];
        int j = working_set[1];

        ++iter;

        // update alpha[i] and alpha[j], handle bounds carefully

        float[] Q_i = Q.get_Q(i,active_size);
        float[] Q_j = Q.get_Q(j,active_size);

        double C_i = get_C(i);
```

```
double C_j = get_C(j);

double old_alpha_i = alpha[i];
double old_alpha_j = alpha[j];

if(y[i]!=y[j])
{
	double quad_coef = QD[i]+QD[j]+2*Q_i[j];
	if (quad_coef <= 0)
		quad_coef = 1e-12;
	double delta = (-G[i]-G[j])/quad_coef;
	double diff = alpha[i] - alpha[j];
	alpha[i] += delta;
	alpha[j] += delta;

	if(diff > 0)
	{
		if(alpha[j] < 0)
		{
			alpha[j] = 0;
			alpha[i] = diff;
		}
	}
	else
	{
		if(alpha[i] < 0)
		{
			alpha[i] = 0;
			alpha[j] = -diff;
		}
	}
	if(diff > C_i - C_j)
	{
		if(alpha[i] > C_i)
		{
			alpha[i] = C_i;
			alpha[j] = C_i - diff;
		}
	}
	else
	{
		if(alpha[j] > C_j)
		{
			alpha[j] = C_j;
			alpha[i] = C_j + diff;
		}
	}
}
else
{
	double quad_coef = QD[i]+QD[j]-2*Q_i[j];
	if (quad_coef <= 0)
		quad_coef = 1e-12;
	double delta = (G[i]-G[j])/quad_coef;
	double sum = alpha[i] + alpha[j];
```

```
		alpha[i] -= delta;
		alpha[j] += delta;

		if(sum > C_i)
		{
			if(alpha[i] > C_i)
			{
				alpha[i] = C_i;
				alpha[j] = sum - C_i;
			}
		}
		else
		{
			if(alpha[j] < 0)
			{
				alpha[j] = 0;
				alpha[i] = sum;
			}
		}
		if(sum > C_j)
		{
			if(alpha[j] > C_j)
			{
				alpha[j] = C_j;
				alpha[i] = sum - C_j;
			}
		}
		else
		{
			if(alpha[i] < 0)
			{
				alpha[i] = 0;
				alpha[j] = sum;
			}
		}
	}

	// update G

	double delta_alpha_i = alpha[i] - old_alpha_i;
	double delta_alpha_j = alpha[j] - old_alpha_j;

	for(int k=0;k<active_size;k++)
	{
		G[k] += Q_i[k]*delta_alpha_i + Q_j[k]*delta_alpha_j;
	}

	// update alpha_status and G_bar

	{
		boolean ui = is_upper_bound(i);
		boolean uj = is_upper_bound(j);
		update_alpha_status(i);
		update_alpha_status(j);
		int k;
```

```
                if(ui != is_upper_bound(i))
                {
                        Q_i = Q.get_Q(i,l);
                        if(ui)
                                for(k=0;k<l;k++)
                                        G_bar[k] -= C_i * Q_i[k];
                        else
                                for(k=0;k<l;k++)
                                        G_bar[k] += C_i * Q_i[k];
                }

                if(uj != is_upper_bound(j))
                {
                        Q_j = Q.get_Q(j,l);
                        if(uj)
                                for(k=0;k<l;k++)
                                        G_bar[k] -= C_j * Q_j[k];
                        else
                                for(k=0;k<l;k++)
                                        G_bar[k] += C_j * Q_j[k];
                }
        }

}

if(iter >= max_iter)
{
        if(active_size < l)
        {
        // reconstruct the whole gradient to calculate objective value
                reconstruct_gradient();
                active_size = l;
                svm.info("*");
        }
        System.err.print(
        "\nWARNING: reaching max number of iterations\n");
}

// calculate rho

si.rho = calculate_rho();

// calculate objective value
{
        double v = 0;
        int i;
        for(i=0;i<l;i++)
                v += alpha[i] * (G[i] + p[i]);

        si.obj = v/2;
}

// put back the solution
{
        for(int i=0;i<l;i++)
```

```
                alpha_[active_set[i]] = alpha[i];
        }

        si.upper_bound_p = Cp;
        si.upper_bound_n = Cn;

        svm.info("\noptimization finished, #iter = "+iter+"\n");
}

// return 1 if already optimal, return 0 otherwise
int select_working_set(int[] working_set)
{
        // return i,j such that
        // i: maximizes -y_i * grad(f)_i, i in I_up(\alpha)
        // j: mimimizes the decrease of obj value
        //    (if quadratic coefficeint <= 0, replace it with tau)
        //    -y_j*grad(f)_j < -y_i*grad(f)_i, j in I_low(\alpha)

        double Gmax = -INF;
        double Gmax2 = -INF;
        int Gmax_idx = -1;
        int Gmin_idx = -1;
        double obj_diff_min = INF;

        for(int t=0;t<active_size;t++)
                if(y[t]==+1)
                {
                        if(!is_upper_bound(t))
                                if(-G[t] >= Gmax)
                                {
                                        Gmax = -G[t];
                                        Gmax_idx = t;
                                }
                }
                else
                {
                        if(!is_lower_bound(t))
                                if(G[t] >= Gmax)
                                {
                                        Gmax = G[t];
                                        Gmax_idx = t;
                                }
                }

        int i = Gmax_idx;
        float[] Q_i = null;
        if(i != -1) // null Q_i not accessed: Gmax=-INF if i=-1
                Q_i = Q.get_Q(i,active_size);

        for(int j=0;j<active_size;j++)
        {
                if(y[j]==+1)
                {
                        if (!is_lower_bound(j))
                        {
```

```
                        double grad_diff=Gmax+G[j];
                        if (G[j] >= Gmax2)
                                Gmax2 = G[j];
                        if (grad_diff > 0)
                        {
                                double obj_diff;
                                double quad_coef = QD[i]+QD[j]-
                                2.0*y[i]*Q_i[j];
                                if (quad_coef > 0)
                                        obj_diff = -
                                (grad_diff*grad_diff)/quad_coef;
                                else
                                        obj_diff = -(grad_diff*grad_diff)/
                                        1e-12;

                                if (obj_diff <= obj_diff_min)
                                {
                                        Gmin_idx=j;
                                        obj_diff_min = obj_diff;
                                }
                        }
                }
        }
        else
        {
                if (!is_upper_bound(j))
                {
                        double grad_diff= Gmax-G[j];
                        if (-G[j] >= Gmax2)
                                Gmax2 = -G[j];
                        if (grad_diff > 0)
                        {
                                double obj_diff;
                                double quad_coef =
                                QD[i]+QD[j]+2.0*y[i]*Q_i[j];
                                if (quad_coef > 0)
                                        obj_diff = -
                                        (grad_diff*grad_diff)/quad_coef;
                                else
                                        obj_diff = -(grad_diff*grad_diff)/
                                        1e-12;

                                if (obj_diff <= obj_diff_min)
                                {
                                        Gmin_idx=j;
                                        obj_diff_min = obj_diff;
                                }
                        }
                }
        }
}

if(Gmax+Gmax2 < eps)
        return 1;
```

```
	working_set[0] = Gmax_idx;
	working_set[1] = Gmin_idx;
	return 0;
}

private boolean be_shrunk(int i, double Gmax1, double Gmax2)
{
	if(is_upper_bound(i))
	{
		if(y[i]==+1)
			return(-G[i] > Gmax1);
		else
			return(-G[i] > Gmax2);
	}
	else if(is_lower_bound(i))
	{
		if(y[i]==+1)
			return(G[i] > Gmax2);
		else
			return(G[i] > Gmax1);
	}
	else
		return(false);
}

void do_shrinking()
{
	int i;
	double Gmax1 = -INF;// max { -y_i * grad(f)_i | i in I_up(\alpha) }
	double Gmax2 = -INF;// max { y_i * grad(f)_i | i in I_low(\alpha) }

	// find maximal violating pair first
	for(i=0;i<active_size;i++)
	{
		if(y[i]==+1)
		{
			if(!is_upper_bound(i))
			{
				if(-G[i] >= Gmax1)
					Gmax1 = -G[i];
			}
			if(!is_lower_bound(i))
			{
				if(G[i] >= Gmax2)
					Gmax2 = G[i];
			}
		}
		else
		{
			if(!is_upper_bound(i))
			{
				if(-G[i] >= Gmax2)
					Gmax2 = -G[i];
			}
			if(!is_lower_bound(i))
```

```
                        {
                                if(G[i] >= Gmax1)
                                        Gmax1 = G[i];
                        }
                }
        }

        if(unshrink == false && Gmax1 + Gmax2 <= eps*10)
        {
                unshrink = true;
                reconstruct_gradient();
                active_size = l;
        }

        for(i=0;i<active_size;i++)
                if (be_shrunk(i, Gmax1, Gmax2))
                {
                        active_size--;
                        while (active_size > i)
                        {
                                if (!be_shrunk(active_size, Gmax1, Gmax2))
                                {
                                        swap_index(i,active_size);
                                        break;
                                }
                                active_size--;
                        }
                }
}

double calculate_rho()
{
        double r;
        int nr_free = 0;
        double ub = INF, lb = -INF, sum_free = 0;
        for(int i=0;i<active_size;i++)
        {
                double yG = y[i]*G[i];

                if(is_lower_bound(i))
                {
                        if(y[i] > 0)
                                ub = Math.min(ub,yG);
                        else
                                lb = Math.max(lb,yG);
                }
                else if(is_upper_bound(i))
                {
                        if(y[i] < 0)
                                ub = Math.min(ub,yG);
                        else
                                lb = Math.max(lb,yG);
                }
                else
                {
```

```
                        ++nr_free;
                        sum_free += yG;
                    }
                }

                if(nr_free>0)
                    r = sum_free/nr_free;
                else
                    r = (ub+lb)/2;

                return r;
            }

}

//
// Solver for nu-svm classification and regression
//
// additional constraint: e^T \alpha = constant
//
final class Solver_NU extends Solver
{
    private SolutionInfo si;

    void Solve(int l, QMatrix Q, double[] p, byte[] y,
               double[] alpha, double Cp, double Cn, double eps,
               SolutionInfo si, int shrinking)
    {
        this.si = si;
        super.Solve(l,Q,p,y,alpha,Cp,Cn,eps,si,shrinking);
    }

    // return 1 if already optimal, return 0 otherwise
    int select_working_set(int[] working_set)
    {
        // return i,j such that y_i = y_j and
        // i: maximizes -y_i * grad(f)_i, i in I_up(\alpha)
        // j: minimizes the decrease of obj value
        //    (if quadratic coefficeint <= 0, replace it with tau)
        //    -y_j*grad(f)_j < -y_i*grad(f)_i, j in I_low(\alpha)

        double Gmaxp = -INF;
        double Gmaxp2 = -INF;
        int Gmaxp_idx = -1;

        double Gmaxn = -INF;
        double Gmaxn2 = -INF;
        int Gmaxn_idx = -1;

        int Gmin_idx = -1;
        double obj_diff_min = INF;

        for(int t=0;t<active_size;t++)
            if(y[t]==+1)
            {
```

```
                if(!is_upper_bound(t))
                        if(-G[t] >= Gmaxp)
                        {
                                Gmaxp = -G[t];
                                Gmaxp_idx = t;
                        }
        }
        else
        {
                if(!is_lower_bound(t))
                        if(G[t] >= Gmaxn)
                        {
                                Gmaxn = G[t];
                                Gmaxn_idx = t;
                        }
        }

int ip = Gmaxp_idx;
int in = Gmaxn_idx;
float[] Q_ip = null;
float[] Q_in = null;
if(ip != -1) // null Q_ip not accessed: Gmaxp=-INF if ip=-1
        Q_ip = Q.get_Q(ip,active_size);
if(in != -1)
        Q_in = Q.get_Q(in,active_size);

for(int j=0;j<active_size;j++)
{
        if(y[j]==+1)
        {
                if (!is_lower_bound(j))
                {
                        double grad_diff=Gmaxp+G[j];
                        if (G[j] >= Gmaxp2)
                                Gmaxp2 = G[j];
                        if (grad_diff > 0)
                        {
                                double obj_diff;
                                double quad_coef = QD[ip]+QD[j]-2*Q_ip[j];
                                if (quad_coef > 0)
                                        obj_diff = -
                                        (grad_diff*grad_diff)/quad_coef;
                                else
                                        obj_diff = -(grad_diff*grad_diff)
                                        /1e-12;

                                if (obj_diff <= obj_diff_min)
                                {
                                        Gmin_idx=j;
                                        obj_diff_min = obj_diff;
                                }
                        }
                }
        }
        else
```

```
		{
			if (!is_upper_bound(j))
			{
				double grad_diff=Gmaxn-G[j];
				if (-G[j] >= Gmaxn2)
					Gmaxn2 = -G[j];
				if (grad_diff > 0)
				{
					double obj_diff;
					double quad_coef = QD[in]+QD[j]-2*Q_in[j];
					if (quad_coef > 0)
						obj_diff = -
						(grad_diff*grad_diff)/quad_coef;
					else
						obj_diff = -(grad_diff*grad_diff)/
						1e-12;

					if (obj_diff <= obj_diff_min)
					{
						Gmin_idx=j;
						obj_diff_min = obj_diff;
					}
				}
			}
		}

		if(Math.max(Gmaxp+Gmaxp2,Gmaxn+Gmaxn2) < eps)
			return 1;

		if(y[Gmin_idx] == +1)
			working_set[0] = Gmaxp_idx;
		else
			working_set[0] = Gmaxn_idx;
		working_set[1] = Gmin_idx;

		return 0;
	}

	private boolean be_shrunk(int i, double Gmax1, double Gmax2, double Gmax3,
	double Gmax4)
	{
		if(is_upper_bound(i))
		{
			if(y[i]==+1)
				return(-G[i] > Gmax1);
			else
				return(-G[i] > Gmax4);
		}
		else if(is_lower_bound(i))
		{
			if(y[i]==+1)
				return(G[i] > Gmax2);
			else
				return(G[i] > Gmax3);
```

```
	}
	else
		return(false);
}

void do_shrinking()
{
	double Gmax1 = -INF;
	// max { -y_i * grad(f)_i | y_i = +1, i in I_up(\alpha) }
	double Gmax2 = -INF;
	// max { y_i * grad(f)_i | y_i = +1, i in I_low(\alpha) }
	double Gmax3 = -INF;
	// max { -y_i * grad(f)_i | y_i = -1, i in I_up(\alpha) }
	double Gmax4 = -INF;
	// max { y_i * grad(f)_i | y_i = -1, i in I_low(\alpha) }

	// find maximal violating pair first
	int i;
	for(i=0;i<active_size;i++)
	{
		if(!is_upper_bound(i))
		{
			if(y[i]==+1)
			{
				if(-G[i] > Gmax1) Gmax1 = -G[i];
			}
			else	if(-G[i] > Gmax4) Gmax4 = -G[i];
		}
		if(!is_lower_bound(i))
		{
			if(y[i]==+1)
			{
				if(G[i] > Gmax2) Gmax2 = G[i];
			}
			else	if(G[i] > Gmax3) Gmax3 = G[i];
		}
	}

	if(unshrink == false && Math.max(Gmax1+Gmax2,Gmax3+Gmax4) <= eps*10)
	{
		unshrink = true;
		reconstruct_gradient();
		active_size = l;
	}

	for(i=0;i<active_size;i++)
		if (be_shrunk(i, Gmax1, Gmax2, Gmax3, Gmax4))
		{
			active_size--;
			while (active_size > i)
			{
				if (!be_shrunk(active_size, Gmax1, Gmax2, Gmax3,
				Gmax4))
				{
					swap_index(i,active_size);
```

```
                                break;
                        }
                        active_size--;
                }
        }
}

double calculate_rho()
{
        int nr_free1 = 0,nr_free2 = 0;
        double ub1 = INF, ub2 = INF;
        double lb1 = -INF, lb2 = -INF;
        double sum_free1 = 0, sum_free2 = 0;

        for(int i=0;i<active_size;i++)
        {
                if(y[i]==+1)
                {
                        if(is_lower_bound(i))
                                ub1 = Math.min(ub1,G[i]);
                        else if(is_upper_bound(i))
                                lb1 = Math.max(lb1,G[i]);
                        else
                        {
                                ++nr_free1;
                                sum_free1 += G[i];
                        }
                }
                else
                {
                        if(is_lower_bound(i))
                                ub2 = Math.min(ub2,G[i]);
                        else if(is_upper_bound(i))
                                lb2 = Math.max(lb2,G[i]);
                        else
                        {
                                ++nr_free2;
                                sum_free2 += G[i];
                        }
                }
        }

        double r1,r2;
        if(nr_free1 > 0)
                r1 = sum_free1/nr_free1;
        else
                r1 = (ub1+lb1)/2;

        if(nr_free2 > 0)
                r2 = sum_free2/nr_free2;
        else
                r2 = (ub2+lb2)/2;

        si.r = (r1+r2)/2;
        return (r1-r2)/2;
```

```
        }
}

//
// Q matrices for various formulations
//
class SVC_Q extends Kernel
{
        private final byte[] y;
        private final Cache cache;
        private final double[] QD;

        SVC_Q(svm_problem prob, svm_parameter param, byte[] y_)
        {
                super(prob.l, prob.x, param);
                y = (byte[])y_.clone();
                cache = new Cache(prob.l,(long)(param.cache_size*(1<<20)));
                QD = new double[prob.l];
                for(int i=0;i<prob.l;i++)
                        QD[i] = kernel_function(i,i);
        }

        float[] get_Q(int i, int len)
        {
                float[][] data = new float[1][];
                int start, j;
                if((start = cache.get_data(i,data,len)) < len)
                {
                        for(j=start;j<len;j++)
                                data[0][j] = (float)(y[i]*y[j]*kernel_function(i,j));
                }
                return data[0];
        }

        double[] get_QD()
        {
                return QD;
        }

        void swap_index(int i, int j)
        {
                cache.swap_index(i,j);
                super.swap_index(i,j);
                do {byte _=y[i]; y[i]=y[j]; y[j]=_;} while(false);
                do {double _=QD[i]; QD[i]=QD[j]; QD[j]=_;} while(false);
        }
}

class ONE_CLASS_Q extends Kernel
{
        private final Cache cache;
        private final double[] QD;

        ONE_CLASS_Q(svm_problem prob, svm_parameter param)
        {
```

```
                super(prob.l, prob.x, param);
                cache = new Cache(prob.l,(long)(param.cache_size*(1<<20)));
                QD = new double[prob.l];
                for(int i=0;i<prob.l;i++)
                        QD[i] = kernel_function(i,i);
        }

        float[] get_Q(int i, int len)
        {
                float[][] data = new float[1][];
                int start, j;
                if((start = cache.get_data(i,data,len)) < len)
                {
                        for(j=start;j<len;j++)
                                data[0][j] = (float)kernel_function(i,j);
                }
                return data[0];
        }

        double[] get_QD()
        {
                return QD;
        }

        void swap_index(int i, int j)
        {
                cache.swap_index(i,j);
                super.swap_index(i,j);
                do {double _=QD[i]; QD[i]=QD[j]; QD[j]=_;} while(false);
        }
}

class SVR_Q extends Kernel
{
        private final int l;
        private final Cache cache;
        private final byte[] sign;
        private final int[] index;
        private int next_buffer;
        private float[][] buffer;
        private final double[] QD;

        SVR_Q(svm_problem prob, svm_parameter param)
        {
                super(prob.l, prob.x, param);
                l = prob.l;
                cache = new Cache(l,(long)(param.cache_size*(1<<20)));
                QD = new double[2*l];
                sign = new byte[2*l];
                index = new int[2*l];
                for(int k=0;k<l;k++)
                {
                        sign[k] = 1;
                        sign[k+l] = -1;
                        index[k] = k;
```

```
                index[k+l] = k;
                QD[k] = kernel_function(k,k);
                QD[k+l] = QD[k];
            }
            buffer = new float[2][2*l];
            next_buffer = 0;
        }

        void swap_index(int i, int j)
        {
            do {byte _=sign[i]; sign[i]=sign[j]; sign[j]=_;} while(false);
            do {int _=index[i]; index[i]=index[j]; index[j]=_;} while(false);
            do {double _=QD[i]; QD[i]=QD[j]; QD[j]=_;} while(false);
        }

        float[] get_Q(int i, int len)
        {
            float[][] data = new float[1][];
            int j, real_i = index[i];
            if(cache.get_data(real_i,data,l) < l)
            {
                for(j=0;j<l;j++)
                    data[0][j] = (float)kernel_function(real_i,j);
            }

            // reorder and copy
            float buf[] = buffer[next_buffer];
            next_buffer = 1 - next_buffer;
            byte si = sign[i];
            for(j=0;j<len;j++)
                buf[j] = (float) si * sign[j] * data[0][index[j]];
            return buf;
        }

        double[] get_QD()
        {
            return QD;
        }
    }

    public class svm {
        //
        // construct and solve various formulations
        //
        public static final int LIBSVM_VERSION=317;
        public static final Random rand = new Random();

        private static svm_print_interface svm_print_stdout =
            new svm_print_interface()
        {
            public void print(String s)
            {
                System.out.print(s);
                System.out.flush();
            }
```

```
};

private static svm_print_interface svm_print_string = svm_print_stdout;

static void info(String s)
{
	svm_print_string.print(s);
}

private static void solve_c_svc(svm_problem prob, svm_parameter param,
				double[] alpha, Solver.SolutionInfo si,
				double Cp, double Cn)
{
	int l = prob.l;
	double[] minus_ones = new double[l];
	byte[] y = new byte[l];

	int i;

	for(i=0;i<l;i++)
	{
		alpha[i] = 0;
		minus_ones[i] = -1;
		if(prob.y[i] > 0) y[i] = +1; else y[i] = -1;
	}

	Solver s = new Solver();
	s.Solve(l, new SVC_Q(prob,param,y), minus_ones, y,
		alpha, Cp, Cn, param.eps, si, param.shrinking);

	double sum_alpha=0;
	for(i=0;i<l;i++)
		sum_alpha += alpha[i];

	if (Cp==Cn)
		svm.info("nu = "+sum_alpha/(Cp*prob.l)+"\n");

	for(i=0;i<l;i++)
		alpha[i] *= y[i];
}

private static void solve_nu_svc(svm_problem prob, svm_parameter param,
				double[] alpha, Solver.SolutionInfo si)
{
	int i;
	int l = prob.l;
	double nu = param.nu;

	byte[] y = new byte[l];

	for(i=0;i<l;i++)
		if(prob.y[i]>0)
			y[i] = +1;
		else
			y[i] = -1;
```

```
		double sum_pos = nu*l/2;
		double sum_neg = nu*l/2;

		for(i=0;i<l;i++)
			if(y[i] == +1)
			{
				alpha[i] = Math.min(1.0,sum_pos);
				sum_pos -= alpha[i];
			}
			else
			{
				alpha[i] = Math.min(1.0,sum_neg);
				sum_neg -= alpha[i];
			}

		double[] zeros = new double[l];

		for(i=0;i<l;i++)
			zeros[i] = 0;

		Solver_NU s = new Solver_NU();
		s.Solve(l, new SVC_Q(prob,param,y), zeros, y,
			alpha, 1.0, 1.0, param.eps, si, param.shrinking);
		double r = si.r;

		svm.info("C = "+1/r+"\n");

		for(i=0;i<l;i++)
			alpha[i] *= y[i]/r;

		si.rho /= r;
		si.obj /= (r*r);
		si.upper_bound_p = 1/r;
		si.upper_bound_n = 1/r;
	}

	private static void solve_one_class(svm_problem prob, svm_parameter param,
				double[] alpha, Solver.SolutionInfo si)
	{
		int l = prob.l;
		double[] zeros = new double[l];
		byte[] ones = new byte[l];
		int i;

		int n = (int)(param.nu*prob.l);	// # of alpha's at upper bound

		for(i=0;i<n;i++)
			alpha[i] = 1;
		if(n<prob.l)
			alpha[n] = param.nu * prob.l - n;
		for(i=n+1;i<l;i++)
			alpha[i] = 0;

		for(i=0;i<l;i++)
```

```
		{
			zeros[i] = 0;
			ones[i] = 1;
		}

		Solver s = new Solver();
		s.Solve(l, new ONE_CLASS_Q(prob,param), zeros, ones,
			alpha, 1.0, 1.0, param.eps, si, param.shrinking);
	}

	private static void solve_epsilon_svr(svm_problem prob, svm_parameter param,
				double[] alpha, Solver.SolutionInfo si)
	{
		int l = prob.l;
		double[] alpha2 = new double[2*l];
		double[] linear_term = new double[2*l];
		byte[] y = new byte[2*l];
		int i;

		for(i=0;i<l;i++)
		{
			alpha2[i] = 0;
			linear_term[i] = param.p - prob.y[i];
			y[i] = 1;

			alpha2[i+l] = 0;
			linear_term[i+l] = param.p + prob.y[i];
			y[i+l] = -1;
		}

		Solver s = new Solver();
		s.Solve(2*l, new SVR_Q(prob,param), linear_term, y,
			alpha2, param.C, param.C, param.eps, si, param.shrinking);

		double sum_alpha = 0;
		for(i=0;i<l;i++)
		{
			alpha[i] = alpha2[i] - alpha2[i+l];
			sum_alpha += Math.abs(alpha[i]);
		}
		svm.info("nu = "+sum_alpha/(param.C*l)+"\n");
	}

	private static void solve_nu_svr(svm_problem prob, svm_parameter param,
				double[] alpha, Solver.SolutionInfo si)
	{
		int l = prob.l;
		double C = param.C;
		double[] alpha2 = new double[2*l];
		double[] linear_term = new double[2*l];
		byte[] y = new byte[2*l];
		int i;

		double sum = C * param.nu * l / 2;
		for(i=0;i<l;i++)
```

```
		{
			alpha2[i] = alpha2[i+l] = Math.min(sum,C);
			sum -= alpha2[i];

			linear_term[i] = - prob.y[i];
			y[i] = 1;

			linear_term[i+l] = prob.y[i];
			y[i+l] = -1;
		}

		Solver_NU s = new Solver_NU();
		s.Solve(2*l, new SVR_Q(prob,param), linear_term, y,
			alpha2, C, C, param.eps, si, param.shrinking);

		svm.info("epsilon = "+(-si.r)+"\n");

		for(i=0;i<l;i++)
			alpha[i] = alpha2[i] - alpha2[i+l];
	}

	//
	// decision_function
	//
	static class decision_function
	{
		double[] alpha;
		double rho;
	};

	static decision_function svm_train_one(
		svm_problem prob, svm_parameter param,
		double Cp, double Cn)
	{
		double[] alpha = new double[prob.l];
		Solver.SolutionInfo si = new Solver.SolutionInfo();
		switch(param.svm_type)
		{
			case svm_parameter.C_SVC:
				solve_c_svc(prob,param,alpha,si,Cp,Cn);
				break;
			case svm_parameter.NU_SVC:
				solve_nu_svc(prob,param,alpha,si);
				break;
			case svm_parameter.ONE_CLASS:
				solve_one_class(prob,param,alpha,si);
				break;
			case svm_parameter.EPSILON_SVR:
				solve_epsilon_svr(prob,param,alpha,si);
				break;
			case svm_parameter.NU_SVR:
				solve_nu_svr(prob,param,alpha,si);
				break;
		}
```

```
		svm.info("obj = "+si.obj+", rho = "+si.rho+"\n");

		// output SVs

		int nSV = 0;
		int nBSV = 0;
		for(int i=0;i<prob.l;i++)
		{
			if(Math.abs(alpha[i]) > 0)
			{
				++nSV;
				if(prob.y[i] > 0)
				{
					if(Math.abs(alpha[i]) >= si.upper_bound_p)
					++nBSV;
				}
				else
				{
					if(Math.abs(alpha[i]) >= si.upper_bound_n)
						++nBSV;
				}
			}
		}

		svm.info("nSV = "+nSV+", nBSV = "+nBSV+"\n");

		decision_function f = new decision_function();
		f.alpha = alpha;
		f.rho = si.rho;
		return f;
	}

	// Platt's binary SVM Probablistic Output: an improvement from Lin et al.
	private static void sigmoid_train(int l, double[] dec_values, double[] labels,
					  double[] probAB)
	{
		double A, B;
		double prior1=0, prior0 = 0;
		int i;

		for (i=0;i<l;i++)
			if (labels[i] > 0) prior1+=1;
			else prior0+=1;

		int max_iter=100;	// Maximal number of iterations
		double min_step=1e-10;	// Minimal step taken in line search
		double sigma=1e-12; // For numerically strict PD of Hessian
		double eps=1e-5;
		double hiTarget=(prior1+1.0)/(prior1+2.0);
		double loTarget=1/(prior0+2.0);
		double[] t= new double[l];
		double fApB,p,q,h11,h22,h21,g1,g2,det,dA,dB,gd,stepsize;
		double newA,newB,newf,d1,d2;
		int iter;
```

```
// Initial Point and Initial Fun Value
A=0.0; B=Math.log((prior0+1.0)/(prior1+1.0));
double fval = 0.0;

for (i=0;i<l;i++)
{
	if (labels[i]>0) t[i]=hiTarget;
	else t[i]=loTarget;
	fApB = dec_values[i]*A+B;
	if (fApB>=0)
		fval += t[i]*fApB + Math.log(1+Math.exp(-fApB));
	else
		fval += (t[i] - 1)*fApB +Math.log(1+Math.exp(fApB));
}
for (iter=0;iter<max_iter;iter++)
{
	// Update Gradient and Hessian (use H' = H + sigma I)
	h11=sigma; // numerically ensures strict PD
	h22=sigma;
	h21=0.0;g1=0.0;g2=0.0;
	for (i=0;i<l;i++)
	{
		fApB = dec_values[i]*A+B;
		if (fApB >= 0)
		{
			p=Math.exp(-fApB)/(1.0+Math.exp(-fApB));
			q=1.0/(1.0+Math.exp(-fApB));
		}
		else
		{
			p=1.0/(1.0+Math.exp(fApB));
			q=Math.exp(fApB)/(1.0+Math.exp(fApB));
		}
		d2=p*q;
		h11+=dec_values[i]*dec_values[i]*d2;
		h22+=d2;
		h21+=dec_values[i]*d2;
		d1=t[i]-p;
		g1+=dec_values[i]*d1;
		g2+=d1;
	}

	// Stopping Criteria
	if (Math.abs(g1)<eps && Math.abs(g2)<eps)
		break;

	// Finding Newton direction: -inv(H') * g
	det=h11*h22-h21*h21;
	dA=-(h22*g1 - h21 * g2) / det;
	dB=-(-h21*g1+ h11 * g2) / det;
	gd=g1*dA+g2*dB;


	stepsize = 1;		// Line Search
	while (stepsize >= min_step)
```

```
                {
                        newA = A + stepsize * dA;
                        newB = B + stepsize * dB;

                        // New function value
                        newf = 0.0;
                        for (i=0;i<l;i++)
                        {
                                fApB = dec_values[i]*newA+newB;
                                if (fApB >= 0)
                                        newf += t[i]*fApB + Math.log(1+Math.exp(-
fApB));
                                else
                                        newf += (t[i] - 1)*fApB
+Math.log(1+Math.exp(fApB));
                        }
                        // Check sufficient decrease
                        if (newf<fval+0.0001*stepsize*gd)
                        {
                                A=newA;B=newB;fval=newf;
                                break;
                        }
                        else
                                stepsize = stepsize / 2.0;
                }

                if (stepsize < min_step)
                {
                        svm.info(
                        "Line search fails in two-class probability estimates\n");
                        break;
                }
        }

        if (iter>=max_iter)
                svm.info(
                "Reaching maximal iterations in two-class probability
                estimates\n");
        probAB[0]=A;probAB[1]=B;
    }

    private static double sigmoid_predict(double decision_value, double A,
    double B)
    {
        double fApB = decision_value*A+B;
        if (fApB >= 0)
                return Math.exp(-fApB)/(1.0+Math.exp(-fApB));
        else
                return 1.0/(1+Math.exp(fApB)) ;
    }

    // Method 2 from the multiclass_prob paper by Wu, Lin, and Weng
    private static void multiclass_probability(int k, double[][] r, double[] p)
    {
        int t,j;
```

```
        int iter = 0, max_iter=Math.max(100,k);
        double[][] Q=new double[k][k];
        double[] Qp=new double[k];
        double pQp, eps=0.005/k;

        for (t=0;t<k;t++)
        {
               p[t]=1.0/k;  // Valid if k = 1
               Q[t][t]=0;
               for (j=0;j<t;j++)
               {
                      Q[t][t]+=r[j][t]*r[j][t];
                      Q[t][j]=Q[j][t];
               }
               for (j=t+1;j<k;j++)
               {
                      Q[t][t]+=r[j][t]*r[j][t];
                      Q[t][j]=-r[j][t]*r[t][j];
               }
        }
        for (iter=0;iter<max_iter;iter++)
        {
               // stopping condition, recalculate QP,pQP for numerical accuracy
               pQp=0;
               for (t=0;t<k;t++)
               {
                      Qp[t]=0;
                      for (j=0;j<k;j++)
                             Qp[t]+=Q[t][j]*p[j];
                      pQp+=p[t]*Qp[t];
               }
               double max_error=0;
               for (t=0;t<k;t++)
               {
                      double error=Math.abs(Qp[t]-pQp);
                      if (error>max_error)
                             max_error=error;
               }
               if (max_error<eps) break;

               for (t=0;t<k;t++)
               {
                      double diff=(-Qp[t]+pQp)/Q[t][t];
                      p[t]+=diff;
                      pQp=(pQp+diff*(diff*Q[t][t]+2*Qp[t]))/(1+diff)/(1+diff);
                      for (j=0;j<k;j++)
                      {
                             Qp[j]=(Qp[j]+diff*Q[t][j])/(1+diff);
                             p[j]/=(1+diff);
                      }
               }
        }
        if (iter>=max_iter)
               svm.info("Exceeds max_iter in multiclass_prob\n");
}
```

```
// Cross-validation decision values for probability estimates
private static void svm_binary_svc_probability(svm_problem prob,
svm_parameter param, double Cp, double Cn, double[] probAB)
{
	int i;
	int nr_fold = 5;
	int[] perm = new int[prob.l];
	double[] dec_values = new double[prob.l];

	// random shuffle
	for(i=0;i<prob.l;i++) perm[i]=i;
	for(i=0;i<prob.l;i++)
	{
		int j = i+rand.nextInt(prob.l-i);
		do {int _=perm[i]; perm[i]=perm[j]; perm[j]=_;} while(false);
	}
	for(i=0;i<nr_fold;i++)
	{
		int begin = i*prob.l/nr_fold;
		int end = (i+1)*prob.l/nr_fold;
		int j,k;
		svm_problem subprob = new svm_problem();

		subprob.l = prob.l-(end-begin);
		subprob.x = new svm_node[subprob.l][];
		subprob.y = new double[subprob.l];

		k=0;
		for(j=0;j<begin;j++)
		{
			subprob.x[k] = prob.x[perm[j]];
			subprob.y[k] = prob.y[perm[j]];
			++k;
		}
		for(j=end;j<prob.l;j++)
		{
			subprob.x[k] = prob.x[perm[j]];
			subprob.y[k] = prob.y[perm[j]];
			++k;
		}
		int p_count=0,n_count=0;
		for(j=0;j<k;j++)
			if(subprob.y[j]>0)
				p_count++;
			else
				n_count++;

		if(p_count==0 && n_count==0)
			for(j=begin;j<end;j++)
				dec_values[perm[j]] = 0;
		else if(p_count > 0 && n_count == 0)
			for(j=begin;j<end;j++)
				dec_values[perm[j]] = 1;
		else if(p_count == 0 && n_count > 0)
```

```
                        for(j=begin;j<end;j++)
                                dec_values[perm[j]] = -1;
                else
                {
                        svm_parameter subparam = (svm_parameter)param.clone();
                        subparam.probability=0;
                        subparam.C=1.0;
                        subparam.nr_weight=2;
                        subparam.weight_label = new int[2];
                        subparam.weight = new double[2];
                        subparam.weight_label[0]=+1;
                        subparam.weight_label[1]=-1;
                        subparam.weight[0]=Cp;
                        subparam.weight[1]=Cn;
                        svm_model submodel = svm_train(subprob,subparam);
                        for(j=begin;j<end;j++)
                        {
                                double[] dec_value=new double[1];

                                svm_predict_values(submodel,prob.x[perm[j]],
                                dec_value);
                                dec_values[perm[j]]=dec_value[0];
                                // ensure +1 -1 order; reason not using CV
                                subroutine
                                dec_values[perm[j]] *= submodel.label[0];
                        }
                }
        }
        sigmoid_train(prob.l,dec_values,prob.y,probAB);
}

// Return parameter of a Laplace distribution
private static double svm_svr_probability(svm_problem prob,
svm_parameter param)
{
        int i;
        int nr_fold = 5;
        double[] ymv = new double[prob.l];
        double mae = 0;

        svm_parameter newparam = (svm_parameter)param.clone();
        newparam.probability = 0;
        svm_cross_validation(prob,newparam,nr_fold,ymv);
        for(i=0;i<prob.l;i++)
        {
                ymv[i]=prob.y[i]-ymv[i];
                mae += Math.abs(ymv[i]);
        }
        mae /= prob.l;
        double std=Math.sqrt(2*mae*mae);
        int count=0;
        mae=0;
        for(i=0;i<prob.l;i++)
                if (Math.abs(ymv[i]) > 5*std)
                        count=count+1;
```

```
                else
                        mae+=Math.abs(ymv[i]);
                mae /= (prob.l-count);
                svm.info("Prob. model for test data: target value = predicted value +
z,\nz: Laplace distribution e^(-|z|/sigma)/(2sigma),sigma="+mae+"\n");
                return mae;
        }

        // label: label name, start: begin of each class, count: #data of classes,
        perm: indices to the original data
        // perm, length l, must be allocated before calling this subroutine
        private static void svm_group_classes(svm_problem prob, int[] nr_class_ret,
        int[][] label_ret, int[][] start_ret, int[][] count_ret, int[] perm)
        {
                int l = prob.l;
                int max_nr_class = 16;
                int nr_class = 0;
                int[] label = new int[max_nr_class];
                int[] count = new int[max_nr_class];
                int[] data_label = new int[l];
                int i;

                for(i=0;i<l;i++)
                {
                        int this_label = (int)(prob.y[i]);
                        int j;
                        for(j=0;j<nr_class;j++)
                        {
                                if(this_label == label[j])
                                {
                                        ++count[j];
                                        break;
                                }
                        }
                        data_label[i] = j;
                        if(j == nr_class)
                        {
                                if(nr_class == max_nr_class)
                                {
                                        max_nr_class *= 2;
                                        int[] new_data = new int[max_nr_class];
                                        System.arraycopy(label,0,new_data,0,label.length);
                                        label = new_data;
                                        new_data = new int[max_nr_class];
                                        System.arraycopy(count,0,new_data,0,count.length);
                                        count = new_data;
                                }
                                label[nr_class] = this_label;
                                count[nr_class] = 1;
                                ++nr_class;
                        }
                }

                //
                // Labels are ordered by their first occurrence in the training set.
```

```
        // However, for two-class sets with -1/+1 labels and -1 appears first,
        // we swap labels to ensure that internally the binary SVM has positive
        data corresponding to the +1 instances.
        //
        if (nr_class == 2 && label[0] == -1 && label[1] == +1)
        {
                do {int _=label[0]; label[0]=label[1]; label[1]=_;} while(false);
                do {int _=count[0]; count[0]=count[1]; count[1]=_;} while(false);
                for(i=0;i<l;i++)
                {
                        if(data_label[i] == 0)
                                data_label[i] = 1;
                        else
                                data_label[i] = 0;
                }
        }

        int[] start = new int[nr_class];
        start[0] = 0;
        for(i=1;i<nr_class;i++)
                start[i] = start[i-1]+count[i-1];
        for(i=0;i<l;i++)
        {
                perm[start[data_label[i]]] = i;
                ++start[data_label[i]];
        }
        start[0] = 0;
        for(i=1;i<nr_class;i++)
                start[i] = start[i-1]+count[i-1];

        nr_class_ret[0] = nr_class;
        label_ret[0] = label;
        start_ret[0] = start;
        count_ret[0] = count;
}

//
// Interface functions
//
public static svm_model svm_train(svm_problem prob, svm_parameter param)
{
        svm_model model = new svm_model();
        model.param = param;

        if(param.svm_type == svm_parameter.ONE_CLASS ||
           param.svm_type == svm_parameter.EPSILON_SVR ||
           param.svm_type == svm_parameter.NU_SVR)
        {
                // regression or one-class-svm
                model.nr_class = 2;
                model.label = null;
                model.nSV = null;
                model.probA = null; model.probB = null;
                model.sv_coef = new double[1][];
```

```
		if(param.probability == 1 &&
		   (param.svm_type == svm_parameter.EPSILON_SVR ||
		    param.svm_type == svm_parameter.NU_SVR))
		{
			model.probA = new double[1];
			model.probA[0] = svm_svr_probability(prob,param);
		}

		decision_function f = svm_train_one(prob,param,0,0);
		model.rho = new double[1];
		model.rho[0] = f.rho;

		int nSV = 0;
		int i;
		for(i=0;i<prob.l;i++)
			if(Math.abs(f.alpha[i]) > 0) ++nSV;
		model.l = nSV;
		model.SV = new svm_node[nSV][];
		model.sv_coef[0] = new double[nSV];
		model.sv_indices = new int[nSV];
		int j = 0;
		for(i=0;i<prob.l;i++)
			if(Math.abs(f.alpha[i]) > 0)
			{
				model.SV[j] = prob.x[i];
				model.sv_coef[0][j] = f.alpha[i];
				model.sv_indices[j] = i+1;
				++j;
			}
	}
	else
	{
		// classification
		int l = prob.l;
		int[] tmp_nr_class = new int[1];
		int[][] tmp_label = new int[1][];
		int[][] tmp_start = new int[1][];
		int[][] tmp_count = new int[1][];
		int[] perm = new int[l];

		// group training data of the same class

		svm_group_classes(prob,tmp_nr_class,tmp_label,
		tmp_start,tmp_count,perm);
		int nr_class = tmp_nr_class[0];
		int[] label = tmp_label[0];
		int[] start = tmp_start[0];
		int[] count = tmp_count[0];

		if(nr_class == 1)
			svm.info(
			"WARNING: training data in only one class. See README
			for details.\n");

		svm_node[][] x = new svm_node[l][];
```

```
int i;
for(i=0;i<l;i++)
	x[i] = prob.x[perm[i]];

// calculate weighted C

double[] weighted_C = new double[nr_class];
for(i=0;i<nr_class;i++)
	weighted_C[i] = param.C;
for(i=0;i<param.nr_weight;i++)
{
	int j;
	for(j=0;j<nr_class;j++)
		if(param.weight_label[i] == label[j])
			break;
	if(j == nr_class)
		System.err.print(
		"WARNING: class label "+param.weight_label[i]+"
		specified in weight is not found\n");
	else
		weighted_C[j] *= param.weight[i];
}

// train k*(k-1)/2 models

boolean[] nonzero = new boolean[l];
for(i=0;i<l;i++)
	nonzero[i] = false;
decision_function[] f = new decision_function[nr_class*
(nr_class-1)/2];

double[] probA=null,probB=null;
if (param.probability == 1)
{
	probA=new double[nr_class*(nr_class-1)/2];
	probB=new double[nr_class*(nr_class-1)/2];
}

int p = 0;
for(i=0;i<nr_class;i++)
	for(int j=i+1;j<nr_class;j++)
	{
		svm_problem sub_prob = new svm_problem();
		int si = start[i], sj = start[j];
		int ci = count[i], cj = count[j];
		sub_prob.l = ci+cj;
		sub_prob.x = new svm_node[sub_prob.l][];
		sub_prob.y = new double[sub_prob.l];
		int k;
		for(k=0;k<ci;k++)
		{
			sub_prob.x[k] = x[si+k];
			sub_prob.y[k] = +1;
		}
		for(k=0;k<cj;k++)
```

```
                {
                        sub_prob.x[ci+k] = x[sj+k];
                        sub_prob.y[ci+k] = -1;
                }

                if(param.probability == 1)
                {
                        double[] probAB=new double[2];

                svm_binary_svc_probability(sub_prob,param,
                weighted_C[i],weighted_C[j],probAB);
                        probA[p]=probAB[0];
                        probB[p]=probAB[1];
                }

                f[p] = svm_train_one(sub_prob,param,weighted_C[i],
                weighted_C[j]);
                for(k=0;k<ci;k++)
                        if(!nonzero[si+k] && Math.abs
                        (f[p].alpha[k]) > 0)
                                nonzero[si+k] = true;
                for(k=0;k<cj;k++)
                        if(!nonzero[sj+k] &&
                        Math.abs(f[p].alpha[ci+k]) > 0)
                                nonzero[sj+k] = true;
                ++p;
        }

// build output

model.nr_class = nr_class;

model.label = new int[nr_class];
for(i=0;i<nr_class;i++)
        model.label[i] = label[i];

model.rho = new double[nr_class*(nr_class-1)/2];
for(i=0;i<nr_class*(nr_class-1)/2;i++)
        model.rho[i] = f[i].rho;

if(param.probability == 1)
{
        model.probA = new double[nr_class*(nr_class-1)/2];
        model.probB = new double[nr_class*(nr_class-1)/2];
        for(i=0;i<nr_class*(nr_class-1)/2;i++)
        {
                model.probA[i] = probA[i];
                model.probB[i] = probB[i];
        }
}
else
{
        model.probA=null;
        model.probB=null;
}
```

```
int nnz = 0;
int[] nz_count = new int[nr_class];
model.nSV = new int[nr_class];
for(i=0;i<nr_class;i++)
{
	int nSV = 0;
	for(int j=0;j<count[i];j++)
		if(nonzero[start[i]+j])
		{
			++nSV;
			++nnz;
		}
	model.nSV[i] = nSV;
	nz_count[i] = nSV;
}

svm.info("Total nSV = "+nnz+"\n");

model.l = nnz;
model.SV = new svm_node[nnz][];
model.sv_indices = new int[nnz];
p = 0;
for(i=0;i<l;i++)
	if(nonzero[i])
	{
		model.SV[p] = x[i];
		model.sv_indices[p++] = perm[i] + 1;
	}

int[] nz_start = new int[nr_class];
nz_start[0] = 0;
for(i=1;i<nr_class;i++)
	nz_start[i] = nz_start[i-1]+nz_count[i-1];

model.sv_coef = new double[nr_class-1][];
for(i=0;i<nr_class-1;i++)
	model.sv_coef[i] = new double[nnz];

p = 0;
for(i=0;i<nr_class;i++)
	for(int j=i+1;j<nr_class;j++)
	{
		// classifier (i,j): coefficients with
		// i are in sv_coef[j-1][nz_start[i]...],
		// j are in sv_coef[i][nz_start[j]...]

		int si = start[i];
		int sj = start[j];
		int ci = count[i];
		int cj = count[j];

		int q = nz_start[i];
		int k;
		for(k=0;k<ci;k++)
```

```
                                if(nonzero[si+k])
                                        model.sv_coef[j-1][q++] =
                                        f[p].alpha[k];
                                q = nz_start[j];
                                for(k=0;k<cj;k++)
                                        if(nonzero[sj+k])
                                                model.sv_coef[i][q++] =
                                                f[p].alpha[ci+k];
                                ++p;
                        }
                }
                return model;
        }

        // Stratified cross validation
        public static void svm_cross_validation(svm_problem prob, svm_parameter param,
        int nr_fold, double[] target)
        {
                int i;
                int[] fold_start = new int[nr_fold+1];
                int l = prob.l;
                int[] perm = new int[l];

                // stratified cv may not give leave-one-out rate
                // Each class to l folds -> some folds may have zero elements
                if((param.svm_type == svm_parameter.C_SVC ||
                    param.svm_type == svm_parameter.NU_SVC) && nr_fold < l)
                {
                        int[] tmp_nr_class = new int[1];
                        int[][] tmp_label = new int[1][];
                        int[][] tmp_start = new int[1][];
                        int[][] tmp_count = new int[1][];


        svm_group_classes(prob,tmp_nr_class,tmp_label,tmp_start,tmp_count,perm);

                        int nr_class = tmp_nr_class[0];
                        int[] start = tmp_start[0];
                        int[] count = tmp_count[0];

                        // random shuffle and then data grouped by fold
                        using the array perm
                        int[] fold_count = new int[nr_fold];
                        int c;
                        int[] index = new int[l];
                        for(i=0;i<l;i++)
                                index[i]=perm[i];
                        for (c=0; c<nr_class; c++)
                                for(i=0;i<count[c];i++)
                                {
                                        int j = i+rand.nextInt(count[c]-i);
                                        do {int _=index[start[c]+j];
                                                index[start[c]+j]=index[start[c]+i];
                                                index[start[c]+i]=_;} while(false);
                                }
```

```
		for(i=0;i<nr_fold;i++)
		{
			fold_count[i] = 0;
			for (c=0; c<nr_class;c++)
				fold_count[i]+=(i+1)*count[c]/
				nr_fold-i*count[c]/nr_fold;
		}
		fold_start[0]=0;
		for (i=1;i<=nr_fold;i++)
			fold_start[i] = fold_start[i-1]+fold_count[i-1];
		for (c=0; c<nr_class;c++)
			for(i=0;i<nr_fold;i++)
			{
				int begin = start[c]+i*count[c]/nr_fold;
				int end = start[c]+(i+1)*count[c]/nr_fold;
				for(int j=begin;j<end;j++)
				{
					perm[fold_start[i]] = index[j];
					fold_start[i]++;
				}
			}
		fold_start[0]=0;
		for (i=1;i<=nr_fold;i++)
			fold_start[i] = fold_start[i-1]+fold_count[i-1];
	}
	else
	{
		for(i=0;i<l;i++) perm[i]=i;
		for(i=0;i<l;i++)
		{
			int j = i+rand.nextInt(l-i);
			do {int _=perm[i]; perm[i]=perm[j]; perm[j]=_;
			} while(false);
		}
		for(i=0;i<=nr_fold;i++)
			fold_start[i]=i*l/nr_fold;
	}

	for(i=0;i<nr_fold;i++)
	{
		int begin = fold_start[i];
		int end = fold_start[i+1];
		int j,k;
		svm_problem subprob = new svm_problem();

		subprob.l = l-(end-begin);
		subprob.x = new svm_node[subprob.l][];
		subprob.y = new double[subprob.l];

		k=0;
		for(j=0;j<begin;j++)
		{
			subprob.x[k] = prob.x[perm[j]];
			subprob.y[k] = prob.y[perm[j]];
			++k;
```

```
                }
                for(j=end;j<l;j++)
                {
                        subprob.x[k] = prob.x[perm[j]];
                        subprob.y[k] = prob.y[perm[j]];
                        ++k;
                }
                svm_model submodel = svm_train(subprob,param);
                if(param.probability==1 &&
                   (param.svm_type == svm_parameter.C_SVC ||
                    param.svm_type == svm_parameter.NU_SVC))
                {
                        double[] prob_estimates=
                        new double[svm_get_nr_class(submodel)];
                        for(j=begin;j<end;j++)
                                target[perm[j]] = svm_predict_probability(submodel,
                                prob.x[perm[j]],prob_estimates);
                }
                else
                        for(j=begin;j<end;j++)
                                target[perm[j]] = svm_predict(submodel,
                                prob.x[perm[j]]);
        }
}

public static int svm_get_svm_type(svm_model model)
{
        return model.param.svm_type;
}

public static int svm_get_nr_class(svm_model model)
{
        return model.nr_class;
}

public static void svm_get_labels(svm_model model, int[] label)
{
        if (model.label != null)
                for(int i=0;i<model.nr_class;i++)
                        label[i] = model.label[i];
}

public static void svm_get_sv_indices(svm_model model, int[] indices)
{
        if (model.sv_indices != null)
                for(int i=0;i<model.l;i++)
                        indices[i] = model.sv_indices[i];
}

public static int svm_get_nr_sv(svm_model model)
{
        return model.l;
}

public static double svm_get_svr_probability(svm_model model)
```

```
	{
		if ((model.param.svm_type == svm_parameter.EPSILON_SVR ||
model.param.svm_type == svm_parameter.NU_SVR) &&
		    model.probA!=null)
		return model.probA[0];
		else
		{
			System.err.print("Model doesn't contain information for SVR
probability inference\n");
			return 0;
		}
	}

	public static double svm_predict_values(svm_model model, svm_node[] x,
double[] dec_values)
	{
		int i;
		if(model.param.svm_type == svm_parameter.ONE_CLASS ||
		   model.param.svm_type == svm_parameter.EPSILON_SVR ||
		   model.param.svm_type == svm_parameter.NU_SVR)
		{
			double[] sv_coef = model.sv_coef[0];
			double sum = 0;
			for(i=0;i<model.l;i++)
				sum += sv_coef[i] * Kernel.k_function(x,
				model.SV[i],model.param);
			sum -= model.rho[0];
			dec_values[0] = sum;

			if(model.param.svm_type == svm_parameter.ONE_CLASS)
				return (sum>0)?1:-1;
			else
				return sum;
		}
		else
		{
			int nr_class = model.nr_class;
			int l = model.l;

			double[] kvalue = new double[l];
			for(i=0;i<l;i++)
				kvalue[i] = Kernel.k_function(x,model.SV[i],model.param);

			int[] start = new int[nr_class];
			start[0] = 0;
			for(i=1;i<nr_class;i++)
				start[i] = start[i-1]+model.nSV[i-1];

			int[] vote = new int[nr_class];
			for(i=0;i<nr_class;i++)
				vote[i] = 0;

			int p=0;
			for(i=0;i<nr_class;i++)
				for(int j=i+1;j<nr_class;j++)
```

```
                        {
                                double sum = 0;
                                int si = start[i];
                                int sj = start[j];
                                int ci = model.nSV[i];
                                int cj = model.nSV[j];

                                int k;
                                double[] coef1 = model.sv_coef[j-1];
                                double[] coef2 = model.sv_coef[i];
                                for(k=0;k<ci;k++)
                                        sum += coef1[si+k] * kvalue[si+k];
                                for(k=0;k<cj;k++)
                                        sum += coef2[sj+k] * kvalue[sj+k];
                                sum -= model.rho[p];
                                dec_values[p] = sum;

                                if(dec_values[p] > 0)
                                        ++vote[i];
                                else
                                        ++vote[j];
                                p++;
                        }

                int vote_max_idx = 0;
                for(i=1;i<nr_class;i++)
                        if(vote[i] > vote[vote_max_idx])
                                vote_max_idx = i;

                return model.label[vote_max_idx];
        }
}

public static double svm_predict(svm_model model, svm_node[] x)
{
        int nr_class = model.nr_class;
        double[] dec_values;
        if(model.param.svm_type == svm_parameter.ONE_CLASS ||
                        model.param.svm_type == svm_parameter.EPSILON_SVR ||
                        model.param.svm_type == svm_parameter.NU_SVR)
                dec_values = new double[1];
        else
                dec_values = new double[nr_class*(nr_class-1)/2];
        double pred_result = svm_predict_values(model, x, dec_values);
        return pred_result;
}

public static double svm_predict_probability(svm_model model, svm_node[] x,
double[] prob_estimates)
{
        if ((model.param.svm_type == svm_parameter.C_SVC ||
        model.param.svm_type == svm_parameter.NU_SVC) &&
            model.probA!=null && model.probB!=null)
        {
                int i;
```

```
                int nr_class = model.nr_class;
                double[] dec_values = new double[nr_class*(nr_class-1)/2];
                svm_predict_values(model, x, dec_values);

                double min_prob=1e-7;
                double[][] pairwise_prob=new double[nr_class][nr_class];

                int k=0;
                for(i=0;i<nr_class;i++)
                        for(int j=i+1;j<nr_class;j++)
                        {
                                pairwise_prob[i][j]=Math.min(Math.max(
                                sigmoid_predict(dec_values[k],model.probA[k],model.p
                                robB[k]),min_prob),1-min_prob);
                                pairwise_prob[j][i]=1-pairwise_prob[i][j];
                                k++;
                        }
                multiclass_probability(nr_class,pairwise_prob,prob_estimates);

                int prob_max_idx = 0;
                for(i=1;i<nr_class;i++)
                        if(prob_estimates[i] > prob_estimates[prob_max_idx])
                                prob_max_idx = i;
                return model.label[prob_max_idx];
        }
        else
                return svm_predict(model, x);
}

static final String svm_type_table[] =
{
        "c_svc","nu_svc","one_class","epsilon_svr","nu_svr",
};

static final String kernel_type_table[]=
{
        "linear","polynomial","rbf","sigmoid","precomputed"
};

public static void svm_save_model(String model_file_name, svm_model model)
throws IOException
{
        DataOutputStream fp = new DataOutputStream(new BufferedOutputStream
                (new FileOutputStream(model_file_name)));

        svm_parameter param = model.param;

        fp.writeBytes("svm_type "+svm_type_table[param.svm_type]+"\n");
        fp.writeBytes("kernel_type "+kernel_type_table[param.kernel_type]+"\n");

        if(param.kernel_type == svm_parameter.POLY)
                fp.writeBytes("degree "+param.degree+"\n");

        if(param.kernel_type == svm_parameter.POLY ||
           param.kernel_type == svm_parameter.RBF ||
```

```
   param.kernel_type == svm_parameter.SIGMOID)
       fp.writeBytes("gamma "+param.gamma+"\n");

if(param.kernel_type == svm_parameter.POLY ||
   param.kernel_type == svm_parameter.SIGMOID)
       fp.writeBytes("coef0 "+param.coef0+"\n");

int nr_class = model.nr_class;
int l = model.l;
fp.writeBytes("nr_class "+nr_class+"\n");
fp.writeBytes("total_sv "+l+"\n");

{
       fp.writeBytes("rho");
       for(int i=0;i<nr_class*(nr_class-1)/2;i++)
              fp.writeBytes(" "+model.rho[i]);
       fp.writeBytes("\n");
}

if(model.label != null)
{
       fp.writeBytes("label");
       for(int i=0;i<nr_class;i++)
              fp.writeBytes(" "+model.label[i]);
       fp.writeBytes("\n");
}

if(model.probA != null) // regression has probA only
{
       fp.writeBytes("probA");
       for(int i=0;i<nr_class*(nr_class-1)/2;i++)
              fp.writeBytes(" "+model.probA[i]);
       fp.writeBytes("\n");
}
if(model.probB != null)
{
       fp.writeBytes("probB");
       for(int i=0;i<nr_class*(nr_class-1)/2;i++)
              fp.writeBytes(" "+model.probB[i]);
       fp.writeBytes("\n");
}

if(model.nSV != null)
{
       fp.writeBytes("nr_sv");
       for(int i=0;i<nr_class;i++)
              fp.writeBytes(" "+model.nSV[i]);
       fp.writeBytes("\n");
}

fp.writeBytes("SV\n");
double[][] sv_coef = model.sv_coef;
svm_node[][] SV = model.SV;

for(int i=0;i<l;i++)
```

```
		{
			for(int j=0;j<nr_class-1;j++)
				fp.writeBytes(sv_coef[j][i]+" ");

			svm_node[] p = SV[i];
			if(param.kernel_type == svm_parameter.PRECOMPUTED)
				fp.writeBytes("0:"+(int)(p[0].value));
			else
				for(int j=0;j<p.length;j++)
					fp.writeBytes(p[j].index+":"+p[j].value+" ");
			fp.writeBytes("\n");
		}

		fp.close();
	}

	private static double atof(String s)
	{
		return Double.valueOf(s).doubleValue();
	}

	private static int atoi(String s)
	{
		return Integer.parseInt(s);
	}

	public static svm_model svm_load_model(String model_file_name)
	throws IOException
	{
		return svm_load_model(new BufferedReader
		(new FileReader(model_file_name)));
	}

	public static svm_model svm_load_model(BufferedReader fp) throws IOException
	{
		// read parameters

		svm_model model = new svm_model();
		svm_parameter param = new svm_parameter();
		model.param = param;
		model.rho = null;
		model.probA = null;
		model.probB = null;
		model.label = null;
		model.nSV = null;

		while(true)
		{
			String cmd = fp.readLine();
			String arg = cmd.substring(cmd.indexOf(' ')+1);

			if(cmd.startsWith("svm_type"))
			{
				int i;
				for(i=0;i<svm_type_table.length;i++)
```

```
		{
			if(arg.indexOf(svm_type_table[i])!=-1)
			{
				param.svm_type=i;
				break;
			}
		}
		if(i == svm_type_table.length)
		{
			System.err.print("unknown svm type.\n");
			return null;
		}
	}
	else if(cmd.startsWith("kernel_type"))
	{
		int i;
		for(i=0;i<kernel_type_table.length;i++)
		{
			if(arg.indexOf(kernel_type_table[i])!=-1)
			{
				param.kernel_type=i;
				break;
			}
		}
		if(i == kernel_type_table.length)
		{
			System.err.print("unknown kernel function.\n");
			return null;
		}
	}
	else if(cmd.startsWith("degree"))
		param.degree = atoi(arg);
	else if(cmd.startsWith("gamma"))
		param.gamma = atof(arg);
	else if(cmd.startsWith("coef0"))
		param.coef0 = atof(arg);
	else if(cmd.startsWith("nr_class"))
		model.nr_class = atoi(arg);
	else if(cmd.startsWith("total_sv"))
		model.l = atoi(arg);
	else if(cmd.startsWith("rho"))
	{
		int n = model.nr_class * (model.nr_class-1)/2;
		model.rho = new double[n];
		StringTokenizer st = new StringTokenizer(arg);
		for(int i=0;i<n;i++)
			model.rho[i] = atof(st.nextToken());
	}
	else if(cmd.startsWith("label"))
	{
		int n = model.nr_class;
		model.label = new int[n];
		StringTokenizer st = new StringTokenizer(arg);
		for(int i=0;i<n;i++)
```

```
                        model.label[i] = atoi(st.nextToken());

                }
                else if(cmd.startsWith("probA"))
                {
                        int n = model.nr_class*(model.nr_class-1)/2;
                        model.probA = new double[n];
                        StringTokenizer st = new StringTokenizer(arg);
                        for(int i=0;i<n;i++)
                                model.probA[i] = atof(st.nextToken());

                }
                else if(cmd.startsWith("probB"))
                {
                        int n = model.nr_class*(model.nr_class-1)/2;
                        model.probB = new double[n];
                        StringTokenizer st = new StringTokenizer(arg);
                        for(int i=0;i<n;i++)
                                model.probB[i] = atof(st.nextToken());

                }
                else if(cmd.startsWith("nr_sv"))
                {
                        int n = model.nr_class;
                        model.nSV = new int[n];
                        StringTokenizer st = new StringTokenizer(arg);
                        for(int i=0;i<n;i++)
                                model.nSV[i] = atoi(st.nextToken());
                }
                else if(cmd.startsWith("SV"))
                {
                        break;
                }
                else
                {
                        System.err.print
                        ("unknown text in model file: ["+cmd+"]\n");
                        return null;
                }
        }

        // read sv_coef and SV

        int m = model.nr_class - 1;
        int l = model.l;
        model.sv_coef = new double[m][l];
        model.SV = new svm_node[l][];

        for(int i=0;i<l;i++)
        {
                String line = fp.readLine();
                StringTokenizer st = new StringTokenizer(line," \t\n\r\f:");

                for(int k=0;k<m;k++)
                        model.sv_coef[k][i] = atof(st.nextToken());
```

```
                int n = st.countTokens()/2;
                model.SV[i] = new svm_node[n];
                for(int j=0;j<n;j++)
                {
                        model.SV[i][j] = new svm_node();
                        model.SV[i][j].index = atoi(st.nextToken());
                        model.SV[i][j].value = atof(st.nextToken());
                }
        }

        fp.close();
        return model;
}

public static String svm_check_parameter(svm_problem prob,
svm_parameter param)
{
        // svm_type

        int svm_type = param.svm_type;
        if(svm_type != svm_parameter.C_SVC &&
           svm_type != svm_parameter.NU_SVC &&
           svm_type != svm_parameter.ONE_CLASS &&
           svm_type != svm_parameter.EPSILON_SVR &&
           svm_type != svm_parameter.NU_SVR)
        return "unknown svm type";

        // kernel_type, degree

        int kernel_type = param.kernel_type;
        if(kernel_type != svm_parameter.LINEAR &&
           kernel_type != svm_parameter.POLY &&
           kernel_type != svm_parameter.RBF &&
           kernel_type != svm_parameter.SIGMOID &&
           kernel_type != svm_parameter.PRECOMPUTED)
                return "unknown kernel type";

        if(param.gamma < 0)
                return "gamma < 0";

        if(param.degree < 0)
                return "degree of polynomial kernel < 0";

        // cache_size,eps,C,nu,p,shrinking

        if(param.cache_size <= 0)
                return "cache_size <= 0";

        if(param.eps <= 0)
                return "eps <= 0";

        if(svm_type == svm_parameter.C_SVC ||
           svm_type == svm_parameter.EPSILON_SVR ||
           svm_type == svm_parameter.NU_SVR)
                if(param.C <= 0)
```

```
                return "C <= 0";

        if(svm_type == svm_parameter.NU_SVC ||
           svm_type == svm_parameter.ONE_CLASS ||
           svm_type == svm_parameter.NU_SVR)
               if(param.nu <= 0 || param.nu > 1)
                       return "nu <= 0 or nu > 1";

        if(svm_type == svm_parameter.EPSILON_SVR)
               if(param.p < 0)
                       return "p < 0";

        if(param.shrinking != 0 &&
           param.shrinking != 1)
               return "shrinking != 0 and shrinking != 1";

        if(param.probability != 0 &&
           param.probability != 1)
               return "probability != 0 and probability != 1";

        if(param.probability == 1 &&
           svm_type == svm_parameter.ONE_CLASS)
               return "one-class SVM probability output not supported yet";

        // check whether nu-svc is feasible

        if(svm_type == svm_parameter.NU_SVC)
        {
               int l = prob.l;
               int max_nr_class = 16;
               int nr_class = 0;
               int[] label = new int[max_nr_class];
               int[] count = new int[max_nr_class];

               int i;
               for(i=0;i<l;i++)
               {
                       int this_label = (int)prob.y[i];
                       int j;
                       for(j=0;j<nr_class;j++)
                               if(this_label == label[j])
                               {
                                       ++count[j];
                                       break;
                               }

                       if(j == nr_class)
                       {
                               if(nr_class == max_nr_class)
                               {
                                       max_nr_class *= 2;
                                       int[] new_data = new int[max_nr_class];
                                               System.arraycopy(label,0,
                                               new_data,0,label.length);
                                       label = new_data;
```

```
                                        new_data = new int[max_nr_class];
                                        System.arraycopy(count,0,
                                        new_data,0,count.length);
                                        count = new_data;
                                }
                                label[nr_class] = this_label;
                                count[nr_class] = 1;
                                ++nr_class;
                        }
                }

                for(i=0;i<nr_class;i++)
                {
                        int n1 = count[i];
                        for(int j=i+1;j<nr_class;j++)
                        {
                                int n2 = count[j];
                                if(param.nu*(n1+n2)/2 > Math.min(n1,n2))
                                        return "specified nu is infeasible";
                        }
                }
        }

        return null;
    }

    public static int svm_check_probability_model(svm_model model)
    {
        if (((model.param.svm_type == svm_parameter.C_SVC ||
        model.param.svm_type == svm_parameter.NU_SVC) &&
        model.probA!=null && model.probB!=null) ||
        ((model.param.svm_type == svm_parameter.EPSILON_SVR ||
        model.param.svm_type == svm_parameter.NU_SVR) &&
         model.probA!=null))
                return 1;
        else
                return 0;
    }

    public static void svm_set_print_string_function(
    svm_print_interface print_func)
    {
        if (print_func == null)
                svm_print_string = svm_print_stdout;
        else
                svm_print_string = print_func;
    }
}
```

XII. ACKNOWLEDGEMENT

I would like to express my gratitude to the following people who has assisted me as I did my SP.

First, to Sir Geoffrey Solano, thank you for having me as your adopted advisee. Thank you for being concerned and always making time for us even if we are not you real advisees. Thank you so much for giving me this SP topic. Without you, I probably still would not be able to finish my SP right now. Thank you for being a very patient, considerate and understanding adviser and teacher to me and my blockmates.

To Ma'am Perl Gasmen, thank you for also making time to answer my emails and concerns about my SP even if you are busy getting your Masters degree right now. And to Ma'am Pia Poblador, thank you for being accommodating despite the short notice consultation.

To Zach Marasigan whose SP is related to mine, thank you for answering all my questions and giving me pointers and references about microarrays. Your advice has been invaluable in my understanding of my SP topic.

To my blockmates and friends, thank you. To my officemates who thought my SP was cool, thank you. And to the few special ones who urged and inspired me more to finish it, thank you very much.

Finally, to my parents and family, who has always been patient, understanding and supportive in everything I do. Sorry for disappointing you or making you wait. You have been my inspiration for finishing this. This is for you.