University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Glioma Brain Cancer Classification Using Microarrays and Support Vector Machines

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Edward Jedwin Reyes

June 2018

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Glioma Brain Cancer Classification Using Microarrays and Support Vector Machines" prepared and submitted by Edward Jedwin Reyes in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

____
Ma. Sheila A. Magboo, M.S.
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*cand.*)	____________	____________
2. Avegail D. Carpio, M.S.	____________	____________
3. Richard Bryann L. Chua, Ph.D. (*cand.*)	____________	____________
4. Perlita E. Gasmen, M.Sc. (*cand.*)	____________	____________
5. Marvin John C. Ignacio, M.Sc. (*cand.*)	____________	____________
6. Vincent Peter C. Magboo, M.D., M.S.	____________	____________
7. Geoffrey A. Solano, Ph.D. (*cand.*)	____________	____________

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

____
Ma. Sheila A. Magboo, M.S.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

____
Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

____
Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

Brain Cancer is a rare type of cancer that is classified based on their cell origin. Recent advancements in bioinformatics analysis enable us to develop molecularly defined classifications within tumors such as using microarrays. This study aims to construct a tool that would predict the existence of a glioma brain cancer using gene expression data, preprocess it, and classify its subtype using Support Vector Machines. This system would help in early detection of the disease and contribute to earlier treatment of the patient.

Keywords: glioma brain cancer, glioma brain cancer classification, microarrays, gene expression data, data normalization, preprocessing, support vector machines

Contents

I. Introduction

A. Background of the Study

Brain tumors are the result of the abnormal growth of cells in tissues of the brain or spinal cord and can either be benign or malignant. Brain cancer is the result of malignant tumors spreading cancer cells throughout the entire body. These cells do not die and instead, divide rapidly at an uncontrollable rate to create more tumors in healthy tissues that can disrupt the functions of the brain [1, 2].

Brain cancer is a rare type of cancer in which only 1.4 % of all cancer cases are diagnosed with it [3]. According to the International Brain Tumor Alliance, statistics involving brain cancer in the Philippines are incomplete as of 2015 [4].

There are many types of brain cancer that are classified based on their cell origin. The most common and aggressive type type being gliomas which begin in the glial tissue and has the following subtypes: astrocytomas that make up for 27 % of the cases of brain cancer in children, glioblastomas which make up 56 % of the cases in adults, and oligodendrogliomas that only make up 6 % but can possibly evolve to glioblastomas [5].

Brain cancer was first identified in 1873 but up to this date, has no definite cure due to the fact that its cause is yet to be proven. Although most victims of this cancer are usually exposed to radiation, such as people working in oil refineries, handlers of jet fuel or chemicals like benzene, chemists, and rubber-factory workers. Whether brain cancer is hereditary is also yet to be proven. There are also claims that aspartame (an artificial sweetener) causes brain cancer [2].

B. Statement of the Problem

Currently, brain cancer classification is based on clinicopathological features only. MRI is the most common diagnostic tool for brain tumors, and once the MRI shows that a tumor is found in the brain, surgeons conduct biopsy to determine the type of brain tumor. Other diagnostic tools include CT scans and PET scans that varies per patient [6]. Unfortunately, these approaches have only been partially successful in gliomas, the most common malignant tumor for adults because they fail to take into consideration underlying molecular lesions [7]. While recent advancements in bioinformatics analysis enable us to develop molecularly defined classifications within tumors such as using microarrays [7], it never implemented the prediction of glioma brain tumors. Furthermore, microarrays have been a subject of criticism as a method due to the influence of factors such as cellular heterogeneity or variability of morphological features, which are difficult to separate from the studied features [8]. This study aims to construct a tool that would predict the existence of a glioma brain cancer using gene expression data from microarrays, preprocess it using methods such as Decimal Scale Normalization to reduce factors that affect their reliability and classify its subtype using Support Vector Machines should a brain tumor exist in the patient. The results of this process can be used as a decision-support tool in diagnosis and in the validation of examinations.

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

1. How can gene expression data from cDNA microarrays be used to predict the brain condition of the patient?

2. How can you reduce the factors that affect a microarray's reliability in identifying gene expression patterns of a particular disease?

3. How the results of the analysis generated?

C. Objectives of the Study

The goal of this study is to construct an automated diagnostic system that would take gene expression profiles as input and predict the presence of a glioma brain cancer and classify which type of glioma is it associated with.

Generally, the study contains the following objectives:

1. To train the system to determine the presence or absence of glioma brain tumor in a patient

2. To train the system to classify the type of glioma

Specifically, it aims to fulfill these objectives using the following steps:

1. To preprocess the data by reducing other factors such as cellular heterogeneity or variability of morphological features, noise and other non-biological factors that may affect the systems accuracy

2. To predict the presence or absence of glioma brain tumor

3. To classify the brain tumor to its corresponding glioma subtype (Astrocytoma, Oligodendroglioma, Glioblastoma)

D. Significance of the Project

This study aims to provide a decision making-tool for neurologists to diagnose glioma brain cancer. This also aims to provide a system these specialists can use to classify the type of glioma the patient has which could greatly contribute to earlier treatment of the patient since gliomas are the most common and most aggressive type of brain cancer.

This study might also aid in determining the appropriate treatment for the patient for each type of glioma contains different cell compositions that might have different ways of being treated efficiently and finally, the study aims to help save lives by using microarray data for early detection.

E. Scope and Limitations

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

1. Input dataset would be composed of 180 gene expression profiles downloaded at ftp://ftp.ncbi.nih.gov/pub/geo/DATA/SOFT/GDS/GDS1962.soft.gz

2. The system would let the user choose which preprocessing method/s the system will use for the given data. If no preprocessing method is chosen, normalization by decimal scaling would be its default preprocessing method.

3. The system predicts the presence or absence of glioma brain cancer based from the classification rule generated after training the data.

4. If the data contains the presence of a glioma brain tumor, it classifies the specific type of glioma present.

F. Assumptions

The following are the assumptions of the following system:

1. Input dataset is already extracted and in numerical format

2. Gene expression profiles of microarrays is considered for input datasets

3. Training data and testing data are in .csv format. The rows would correspond to the gene expression identifiers and the columns would be the glioma brain samples

4. Training and testing data do not contain any missing values

5. The system is to be used by neurologists, brain cancer specialists, and other health care professionals

II. Review of Related Literature

Scientists and researchers have recently been using DNA microarrays in measuring the expression levels of large amounts of genes simultaneously. This process has aided them in classifying diseases and determining the genes that contribute to a particular disease in a parallel, rapid and efficient manner. Cancer in particular has been a popular example for the application of microarrays because cancer classification has been one of the most difficult diseases to classify, mainly due to the fact that tumors can develop in any type of cell [9].

Past studies have shown that classifying these tumors using microarrays is possible and has even aided in identifying different subclasses of the tumors and the genes that contribute to it. Shai et al. used gene expression data to identify three subtypes of gliomas: Lower grade Astrocytomas, Glioblastomas, and Oligodendrogliomas. They used K-means clustering and hierarchical clustering analysis on the microarray data and even determined two subtypes of Glioblastomas. The experiment also indicated that a relatively small number of genes can be used to distinguish between these molecular subtypes [7].

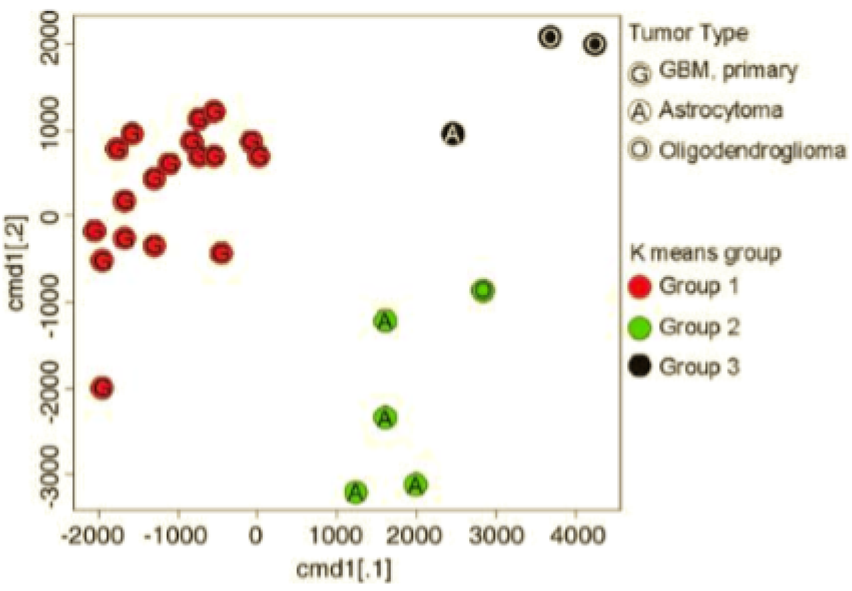


Figure 1: K-means clustering results defining subtypes of gliomas

Nutt et al. classified gliomas using microarrays and concluded that gene-expression-based grouping of tumors is a more powerful survival predictor than histologic grade or age [10]. The experiment demonstrated that gene-expression analysis showed unrecognized heterogeneity of tumors and is efficient in selecting gene expression differences. Frejie et al. also did a similar experiment but compared it to histological classification. They concluded that microarray classification is a more accurate predictor of prognosis than histological classification [11].

Support Vector Machines have also contributed in classifying brain cancer types. Using given labelled training data, the algorithm outputs an optimal hyperplane which categorizes the samples. Ghotekar et al. used Support Vector Machines to detect and classify brain cancer in MRI images and results showed an accuracy of 83.33 % [12]. Bauer et al. also did a similar experiment but added CRF regularization to accelerate computation time. They concluded by using CRF regularization, accuracy is higher for larger amounts of patients [13].

There are also studies that implemented Support Vector Machines in classifying microarray gene expression data. George et al. stated that SVMs achieve the best classification performance compared to other classification methods such as K-nearest neighbors, back propagation neural networks, probabilistic neural networks, weighted voting methods and decision trees because SVMs have demonstrated to not only separate entities into appropriate classes respectively, it also identifies instances whose established classification is not supported by the data [14].

Other applications of SVM in gene expression included Guyon et al. which classified leukemia and colon cancer and showed a better performance than baseline method with a 95.8 % confidence [15]. Furey et al. classified ovarian cancer, leukemia, and colon cancer using SVM and aside for its ability to correctly classify tissue and cell types, it also demonstrated its ability to identify mislabeled data

[16]. Ramaswanny et al. used One-vs-All SVM on classifying tumor tissues and showed an accuracy of 78 % [17]. Lee et al. used Multicategory Support Vector Machines which aims to diagnose multiple cancer types respectively to classify leukemia and small round blue cell tumor data. The experiment resulted in 0 to 1 test error at best [18].

Preprocessing and normalization techniques also play an important role in interpreting the results of classifying microarrays. Normalization reduces the variation between microarrays of non-biological origin. Normalizing data attempts to give all attributes equal weight and it helps prevent attributes with initially large ranges from outweighing attributes with initially smaller ranges [19, 20]. Normalization methods include min-max normalization, z-score normalization, and normalization by decimal scaling [19].

C. Cheadle et al. analyzed microarray data using z-score transformation. They stated that Z scores provide a useful measurement of gene expression and has been used directly on hierarchical clustering, k-means clustering, self-organizing maps, principal component analysis, multidimensional scaling and visualization programs. It formed the basis of comparison of hybridization intensity data among many experiments within the same data type and as such offers a useful method for the basic analysis of microarray data [20].

Bolstad et al. presented another normalization method called quantile normalization so that data from all arrays are used to form the normalizing relation. It was compared to other methods such as scaling normalization and non-linear normalization. They concluded that the quantile method outperforms the other methods in terms of speed, variance and bias and should be used in preference to other methods [21].

III. Theoretical Framework

Definition of Terms

- Brain Cancer/Brain Tumor - result of the abnormal growth of cells in tissues of the brain or spinal cord and can either be benign or malignant [1, 2].
- Primary Brain Cancer - tumors that originated from the brain or the spinal cord [1].
- Glioma - most frequent type of Primary Brain Cancer [1]. Has 3 main types:
  - Astrocytoma - glioma that develops from star-shaped cells that support nerve cells [5]. It is the most common type of brain cancer in children and composes of about 27 % of all cases of brain cancer [1].
  - Oligodendroglioma glioma that develops from glial cells called oligodendrocytes. It is responsible for around 6 % of all brain cancer cases [5].
  - Ependymoma - glioma that develops in the ependymal, or cells that line the passageway of the brain. It is one of the least common glioma cancer types but most cases came from adults aged 20-42 [5].
- Glioblastoma - grade 4 astrocytoma which is the most aggressive type of brain cancer common among adults. It accounts for about 56 % of Brain Cancer cases in adults [5].
- Gene - basic unit of heredity. They are made up of DNA and encode instructions to make protein molecules. Genes are widely used to test a persons susceptibility to cancer and other diseases [22].
- Gene Expression - process where DNA information is converted to mRNA and is translated to proteins [23].
- Deoxyrybonucleic acid (DNA) hereditary material found in almost every organism including humans [24].

- Metastasis the process of spreading cancer cells into other parts of the body [1, 2].

- Biomarker - characteristics that can be observed from a given medical condition or biological state [25].

Microarray

A Microarray is a tool for analyzing gene expression. It contains a glass slide consisting of a large number of genes arranged in a regular pattern and is used to survey these genes quickly or when the sample to be studied is small. Using microarrays, scientists can determine the expression levels of hundreds or thousands of genes in single experiment by measuring the amount of mRNA bound to each site on the array [26].

DNA Microarray

DNA microarrays are small, solid supports onto which the sequences from thousands of different genes are immobilized, or attached, at fixed locations. Their locations are used to identify a particular gene sequence [26]. It is a tool used to determine mutation in genes and is particularly useful in identifying abnormal cells such as tumor cells [27].

To study gene expression patterns, the basic approach is to measure the levels of all mRNAs present in a cell. The underlying assumption is that if a gene is being transcribed to mRNA, then the gene is being expressed. First mRNA is extracted from cells and is converted to cDNA using enzyme reverse transcriptase. Then cDNA is labelled with flourescent markers (typically, tumor cell cDNA as red and normal cell cDNA as green) and are then mixed together and allowed to hybridize to the complementary sequences of DNA of the microarray. As laser scanner then measures the flourescence of each spot seperately for red and green and the results of one color are superimposed over those of the other. This process is called hybridization probing [26, 27].

Machine Learning

Machine learning is a type of Artificial Intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Similar to data mining, it uses extracted data to detect patterns and adjust program actions accordingly [28].

Supervised Machine Learning

Supervised machine learning is when all data is labelled and the algorithms learn to predict the output from the input data. It is the process where the algorithms can apply what has been learned in the past to iteratively make predictions on the training data and is corrected by the teaching data. The learning stops when the algorightm achieves an acceptable level of performance [28, 29].

Statistical Classification

Statistical classification is a method to build predicative models to seperate and classify new data points [30]. It is considered a part of machine learning because the system, or classifier, needs to have a set of already known classified data as a training set to learn the characteristics of the labels and identify which set of categories future unlabelled input belongs [31]. It uses discrete, categorical random variables of interest and has a fixed set of categories [32]. It has two types: binary classification, which involves only two classes and multiclass classification, which involves 3 or more classes [33].

To build a classifier, we first define its attributes in order to classify the data. This could be done by first, stating explicit rules or by defining them through training examples called the training set. In general, the training set is a set of data that is already classified and is used to build the classification model. The training data is defined as X = $\{X_1, X_2, ..., X_n\}$. Then we define the feature space by selecting the features we want to measure. Features can be either discrete or continuous. If m features are measured, each sample would contain a m X 1 row vector X_i, for $i \in \{1,...,N\}$of data, and we refer to the space $\mathbb{R}^m$ as the feature space. The training data is a N x m matrix, where entry X_{ij} represents the j^{th} feature of the i^{th} sample. Next we define the decision algorithm by setting parameters from the training set which would produce a prediction of the sample. Finally we measure its performance by calculating the weighted error rate [30, 32].

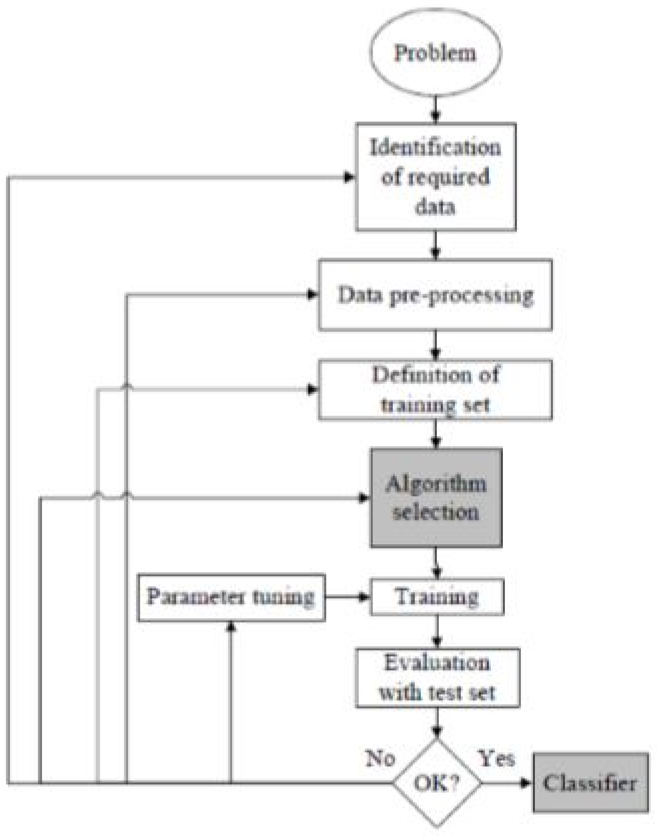


Figure 2. The process of supervised machine learning [33]

Data Normalization and Preprocessing

Data Normalization and Preprocessing is the process of manipulating data to make measurements from multiple arrays comparable [34]. They play an important role in interpreting the results of classifying microarrays. Normalization reduces the variation between microarrays of non-biological origin. Normalizing data attempts to give all attributes equal weight and it helps prevent attributes with initially large ranges from outweighing attributes with initially smaller ranges

[19]. Several microarray data have been proposed in the past, the methods are the following:

Normalization by Decimal Scaling

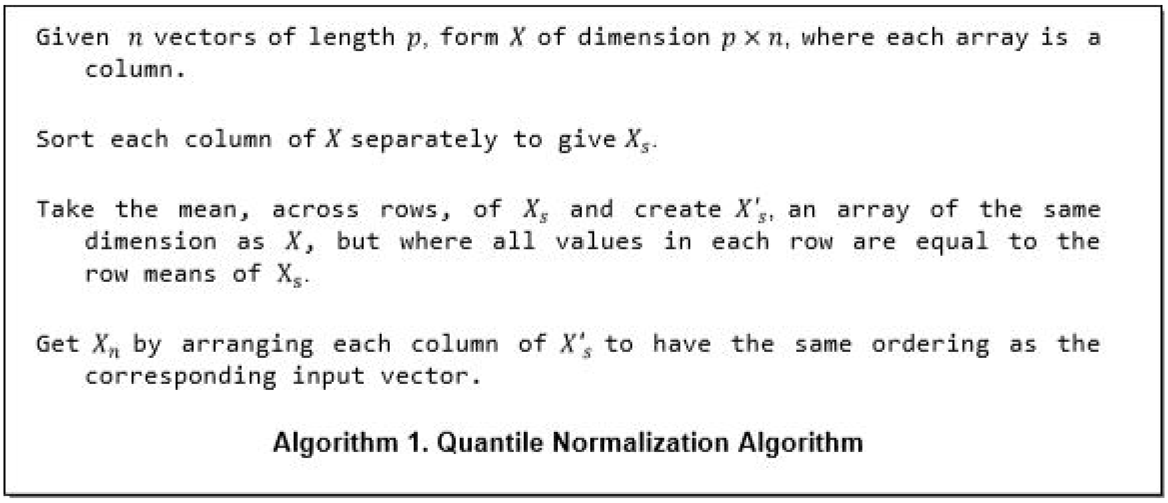
Normalization by decimal scaling normalizes data by moving by moving the decimal point of values of attribute A. The number of decimal points moved depends on the maximum absolute value of A. A value V_i, of A is normalized to v' by computing

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

where j is the smallest integer such that $max(|V'_i| < 1)$ [19].

Quantile Normalization

Quantile method seeks to make the same empirical distribution of probe intensities for each array in a set of arrays by transforming the distribution of intensities from one distribution to another. The method is motivated by the idea that a quantile-quantile plot shows that the distribution of two data vectors is the same if the plot is a straight diagonal line and not the same if it is other than a diagonal line [19, 21]. This implies that we can give each array the same distribution by taking the mean quantile and substituting it as the value of the data item in the original dataset.



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 linear transformation on the original data. Suppose that min_A and max_A are the minimum and maximum values of an attribute, A. Min-max normalization maps a value, v_i, of A to v_i' in the range $[new\_min_A, new\_max_A]$ by computing

$$v_i' = \frac{v_i - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A.$$

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

In z-score transformation (or zero-mean normalization), the values for an attribute, A, are normalized based on the mean and standard deviation of A. A value, v_i of A is normalized to v_i' by computing

$$v_i' = \frac{v_i - \bar{A}}{\sigma_A},$$

where $\bar{A}$ and σ_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 there are outliers that dominate the min-max normalization.

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a discriminative classifier formally defined by a seperating hyperplane. Given labelled training data, the algorithm outputs an optimal hyperplane which categorizes new examples [35]. The seperating hyperplane is defined as

$$w . x + b = o$$

where w is the normal vector and b as the offset. The vector w points perpendicular to the seperating hyperplane and adding the offset parameter b allows us to increase the margin [36]. The margin is the distance to the separating hyperplane and intuitively, a measure of confidence. Therefore, the optimal separating hyperplane maximizes the margin of the training data [34, 35].

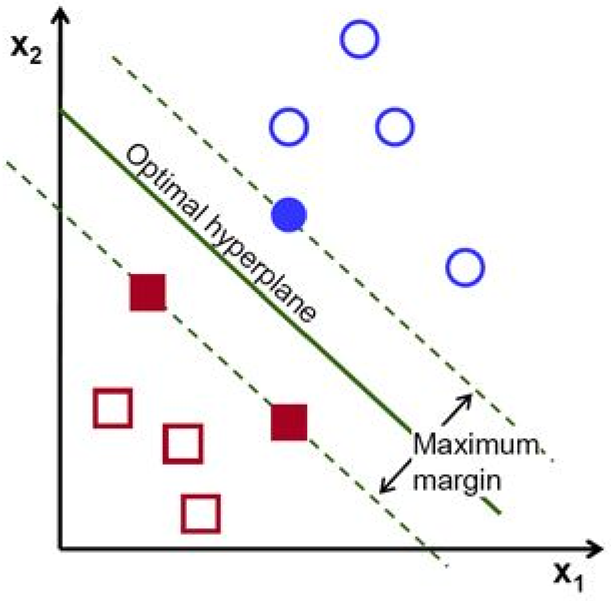


Figure 3. Visualization of Maximal Margin Hyperplane [35]

To find the maximum margin hyperplane, given samples x_i, y_i where $x_i \ \epsilon\ \Re^n, y_i \ \epsilon\ \{-1, +1\}$, we find the hyperplane $v \cdot x = 0$ with $\|v\| = 1$. The margin δ is given as

$$\delta = y(v \cdot x)$$

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

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

In the event that the data is not linearly separable, there are two possible ways to tackle the problem. One is to penalize each point by distance from margin 1, that is, minimize:

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

The result would be complex in low dimensional space, which would lead to a statistical problem called "curse of dimensionality" for the amount of data needed is often proportional to the number of dimensions n mapped to $O(n^d)$ dimensions which would be very expensive in time and memory if it is to be computed [37, 34].

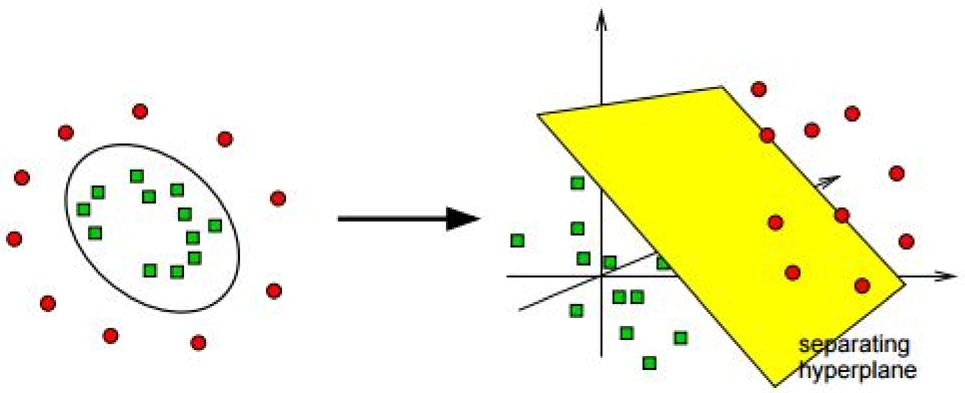


Figure 4. Nonlinear separable data being separable in higher dimensions

The other solution would be to map the data into a higher dimensional space in which it becomes linearly separable. This process is called "kernel trick" [38, 34]. The construction of the maximal margin hyperplane in higher dimensional spaces would depend on its inner products, nonetheless, this solution would still be very efficient in large dimensions.

If classification in higher dimensional planes would be easier, we would want to construct a dividing hyperplane there then calculate the scalar products of the form $\langle \phi(p), \phi(q) \rangle$ which would be very difficult if the dimension becomes too large. So instead of using scalar product, SVM uses a kernel function K which behaves like its inner product [38].

$$\mathcal{K}(p, q) = \langle \ \Phi(p), \Phi(q) \ \rangle$$

The similarity in gene space would then be the inner product defined as

$$\langle p, q \rangle = \sum_{i=1}^{g} p_i q_i = p_1 q_1 + p_2 q_2 + \ldots + p_g q_g$$

The similarity in feature space would be the kernel function. The four basic kernels would be [39, 38]:

$$\text{Linear: } K(p,q) = p^T q$$
$$\text{Polynomial: } K(p,q) = (\gamma p^T q + r)^d, \gamma > 0$$
$$\text{Radial Basis Function (RBF): } K(p,q) = exp(-\gamma \|p-q\|^2), \gamma > 0$$
$$\text{Sigmoid: } K(p,q) = tanh(\gamma p^T q + r)$$

Here, γ (the width of RBF coefficient in polynomial (=1)), d (degree of the polynomial), and r (additive constant in polynomial (=0)) are the parameters of the kernel [34, 39].

In using SVMs in microarray data classification, there are certain rules that need to be taken note of [37]:

1. Data normalization:

   Rescale data such that all kernel values fall between -100 and 100; a simple way to do this is by normalizing all entries of the microarray such that they fall between $-10(n)^{\frac{1}{2}} and + 10(n)^{\frac{1}{2}}$, where n is the number of expression values per sample.

2. Choosing the regularization parameter C:

   Given the above normalization, the regularization parameter usually does not have much effect, so set it somewhere between 1-100.

3. Choosing the kernel:

   For microarray applications, a linear kernel is usually sufficient. Polynomial kernels can be used to examine correlations between genes but they do not greatly improve performance, so the Gaussian kernel can be used as an alternative.

There are many types of SVM kernel functions. The commonly used functions are [40]

1. Linear Kernel:

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

   The Linear Kernel uses data transposition which generates a standard linear classifier. The 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 ϵ $\Re^n + 1$. This is a reasonable choice of kernel as it avoids fitting a very complicated nonlinear function in a very high dimensional feature space (overfitting).

2. Radial Basis Function Kernel [41]:

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

   This kernel function is in Gaussian form, where σ is the width of the Gaussian. If σ^2 is too large, then we will tend to have a higher bias and lower variance classifier, and if σ^2 is too small, then we will tend to have lower bias and higher variance classifier. This Gaussian kernel is used when you have a feature space $\Re^n$ where the number of features n, is very small, and m, the number of training samples, is very large. Feature scaling is needed when when using the Gaussian kernel if the features are on very different scales so that the SVM would give a 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. Using the function, it will generate an array of features and train the SVM from there.

SVM was designed for binary applications, but real clinical applications such as microarray gene classification require multicategory classification methods. In order to address this, we transform the multiclass classification problem into a binary problem. Given a dataset composed of l patterns $(x_1, y_1), ..., (x_l, y_l)$ where

$x_i \epsilon \Re^n$ and $y_i \epsilon$ 1,2,...,k, here are the methods for SVM multiclass classification [42]:

1. One-vs-All method (OVA)

   Each class is learned against the others $(k-1)$, resulting in k classifiers.

2. One-vs-One method (OVO)

   This method builds $\frac{k(k-1)}{2}$ classifiers: each one of them learning the seperation between two classes, where k is a single class.

Performance Evaluation

Several issues need to be addressed in designing a procedure for the assessment of gene expression data such as the proceudre should provide protection against overfitting the data and the time-consuming process of repeatedly fitting high dimensional data. Here are possible methods to evaluate the relative performance of the classification procedures:

1. 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 correclty classified [39].

2. Re-randomization

   Re-randomization involves re-randomizing the entire data and then repeat the modeling and validation steps. This is used because cross-validation may not be sufficient in protecting against overfitting due to the relatively small cross-validated errors achieved by capitalizing on chance properties [43].

IV. Design and Implementation

A. Dataset

The dataset would consist of gene expression profiles for a total of 180 samples. The data includes 23 samples from epilepsy patients (used as non-tumor samples), 26 astrocytoma tumor samples, 50 oligodendroglioma tumor samples, and 81 glioblastoma tumor samples for a total of 157 tumor samples.

The dataset can be used to generate two smaller datasets, a training set and a testing set that contains normal and glioma brain cancer data (Astrocytoma, Oligodendroglioma, and Glioblastoma).

The dataset can be accessed publicly on the NCBI website (ftp://ftp.ncbi.nih.gov/pub/geo/DATA/SOFT/GDS/GDS1962.soft.gz).

The input data of the system should be in a comma delimited format. with rows labelled as the samples and the columns labelled as the genes. The first row of the training dataset should contain the gene code for each gene and each sample should contain a label of the gene classification in string format. The testing set should also contain the gene code for each gene but it doesn't need to contain the label of its gene classification. The same genes would also be used for the training set and the testing set. We will also assume that the dataset does not contain any missing data.

B. System

Generally, the input and output of the system would be defined on the context diagram below:

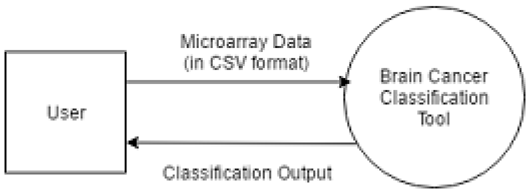

User
Microarray Data (in CSV format)
Brain Cancer Classification Tool
Classification Output

Figure 5. I/O Context Diagram

The system would take gene expression profiles of brain tumor samples as input. The input would be in .csv format. The user can then set the preprocessing methods as parameters. If the user does not set these parameters, normalization by decimal scaling would be its default parameter. A more detailed representation of the system is illustrated below:

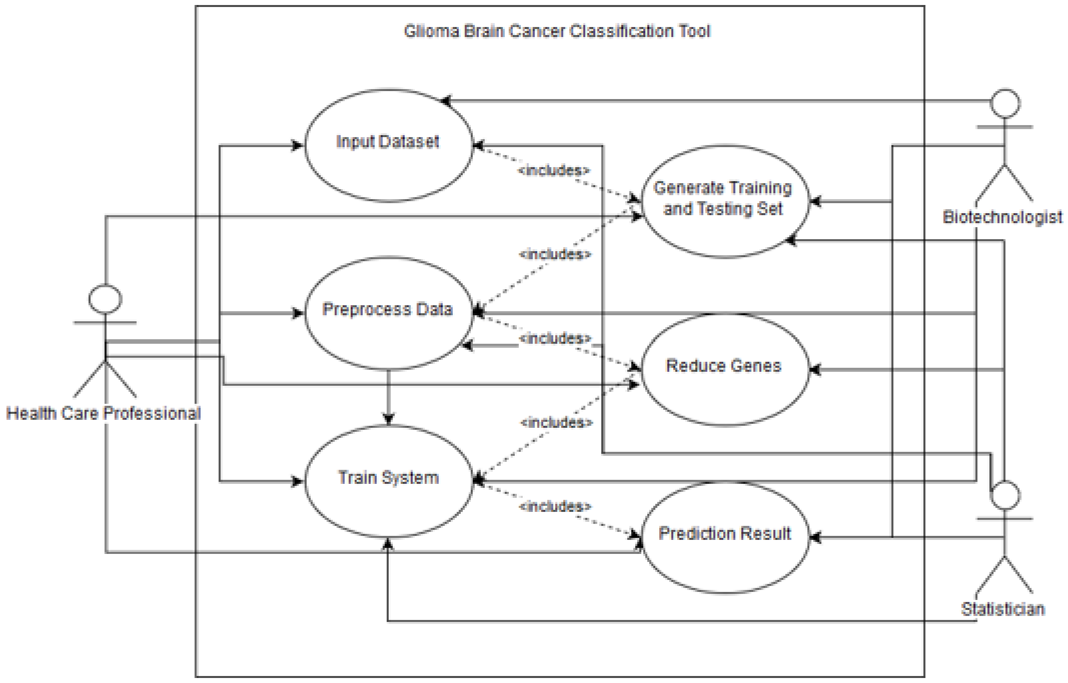


Figure 6. Use Case Diagram

The input data would then be read by the system and the training and testing sets are generated with a 70% - 30% ratio distributed randomly in the data. The data would then undergo preprocessing and normalization to reduce the amount of data by removing non-biological in order to produce more accurate results. If Decimal Scale Normalization is selected, the decimal places of the data would be computed. If Quantile Normalization is selected, the average of the transposed data would be computed. If Z-Score Transformation is selcted, the data's mean intensity and standard deviation would be computed, and if Min-Max Normalization is used, the min and max feature of the data would be identified. The system would then identify which genes contain cancerous attributes, and afterwards, will apply Support Vector Machine to classify which subtype of Glioma Brain Cancer does it belong to. The top level flow of the process is shown below:

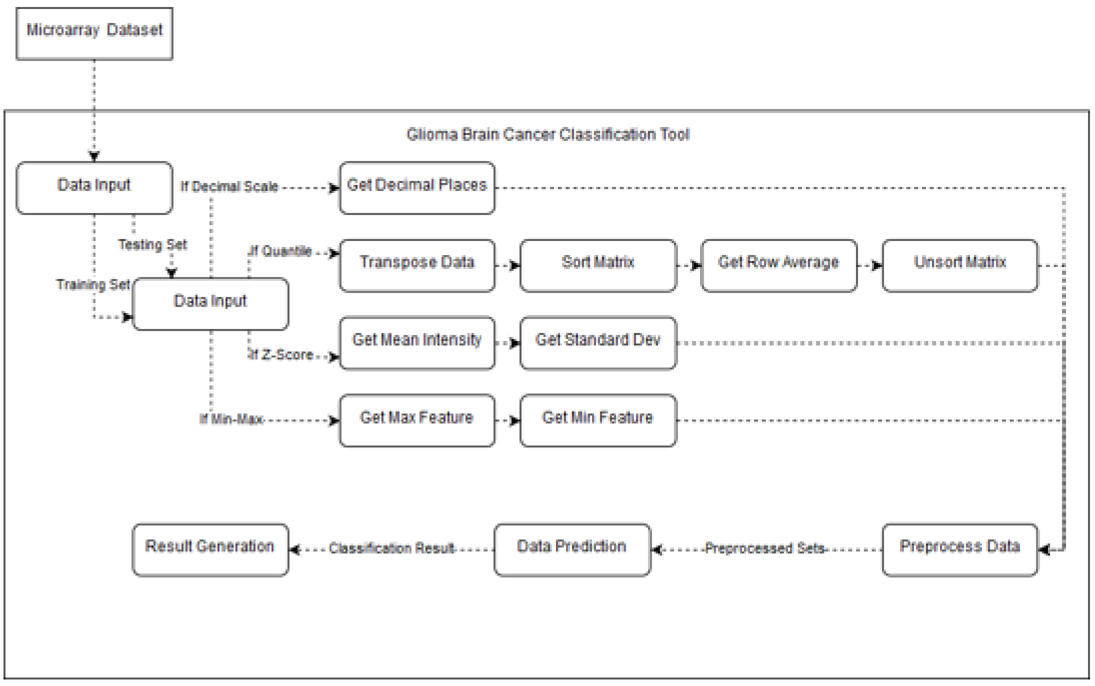


Figure 7. Top Level Data Flow Diagram

If the system has not been trained, the user must input a microarray dataset that is comma delimited and is in .csv format before it can generate training and testing sets and analyze the data. The dataset should also include its disease state or class name such as "non-tumor", "astrocytomas", "glioblastomas", and "oligodendrogliomas".

After the dataset and its parameters are given, the system parses the file to collect the data matrix. It will then divide it into the training set and the testing set with a 70% - 30% ratio. The system selects the sample genes using random sampling without replacement then proceeds to place it into the divided datasets. The process is shown below:

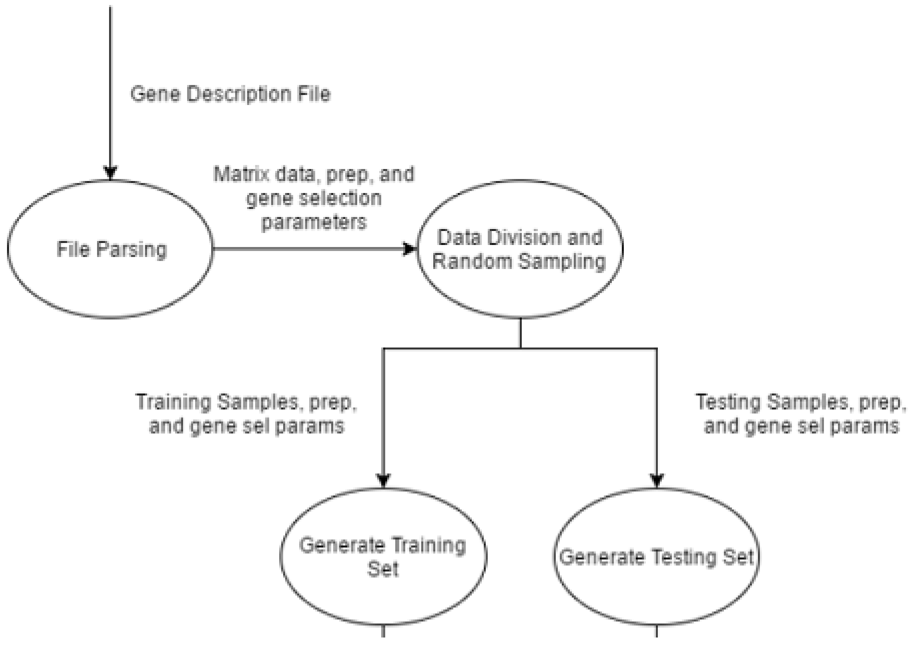

Gene Description File
File Parsing
Matrix data, prep, and gene selection parameters
Data Division and Random Sampling
Training Samples, prep, and gene sel params
Testing Samples, prep, and gene sel params
Generate Training Set
Generate Testing Set

Figure 8. Process Flow 1

The system would then begin to preprocess the data matrices based on the method selected by the system. The user may select which preprocessing method would the system use on the input dataset. The possible preprocessing methods include Normalization by Decimal Scaling, Quantile Normalization, Z-Score Transformation, and Min-Max normalization. If the user does not select a preprocessing method, the system would choose Normalization by Decimal Scaling by default. Preprocessing involves normalizing and transforming the data to remove non-biological factors in the input. This step results to preprocessed data that will be used for dimension reduction and gene selection as shown below:

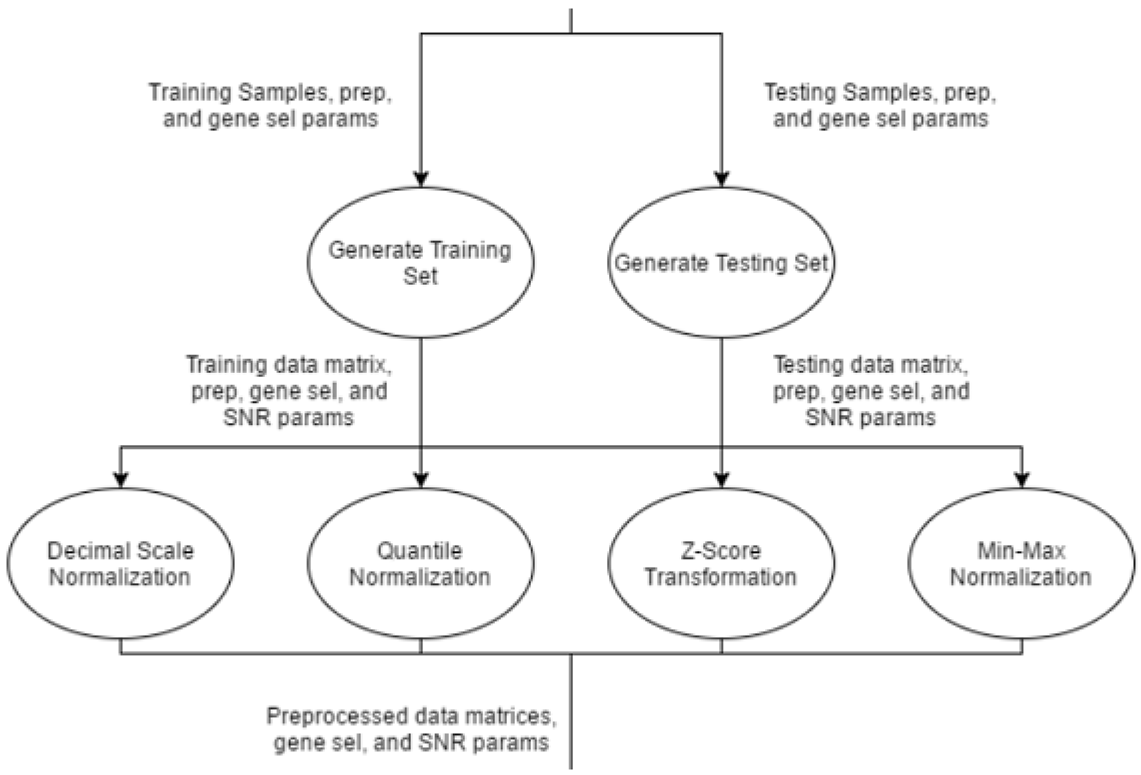

Training Samples, prep, and gene sel params
Testing Samples, prep, and gene sel params
Generate Training Set
Generate Testing Set
Training data matrix, prep, gene sel, and SNR params
Testing data matrix, prep, gene sel, and SNR params
Decimal Scale Normalization
Quantile Normalization
Z-Score Transformation
Min-Max Normalization
Preprocessed data matrices, gene sel, and SNR params

Figure 9. Process Flow 2

The reduced dataset would then undergo microarray dataset analysis which involves training the system and generating the classification rules in order to predict the presence or absence of Glioma Brain Cancer. The system would build a classifier model from the training set for predicting Astrocytoma, Glioblastoma, and Oligodendroglioma. The classification rule generated would then be used in determining the subtype for Glioma Brain Cancer shown in the figure below:

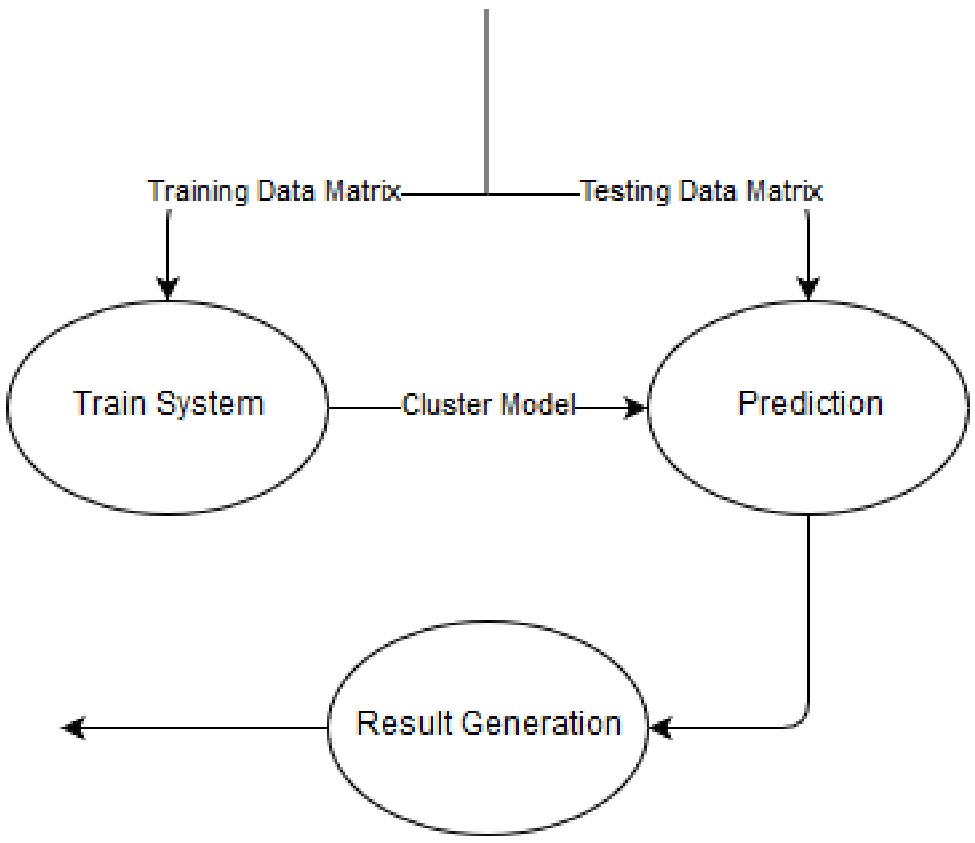


Figure 10. Process Flow 3

The classification result would then be used as the output to the user. The output would consist of the probably Glioma Brain condition of the patient based on the gene expression results. Performance measures and statistics would also be outputted as well as the values computed along the process should the user opt to include it.

SVM Implementation

Support Vector Machines is implemented using a library in R [44]. The chosen regularization parameter is any integer between 1 and 100 [45]. Linear kernel would be used as the training datasets have a small sample size and large number of features. It also supports multiclass clasification using One-vs-one method and will be used for the two multiclass classifiers required by the system.

C. System Architecture

The system is built using the following:

- Eclipse Oxygen 3a ver 4.7.3
- Java Development Kit (1.8.1_71)
- Windows 10 Operating system

The minimum system requirements for an effective run of the system are as follows:

- Java Runtime Enviroment (JRE)
- 8 GB RAM
- 1 GB free hard disk space
- Dual core processor (2.00GHz and 2.50GHz)

V. Results

The main menu consists of mainly the side panel where you can place your input and configure the preprocessing methods and the results panel which would be filled once the application has finished running. The main menu shown below labelled as Figure 11.

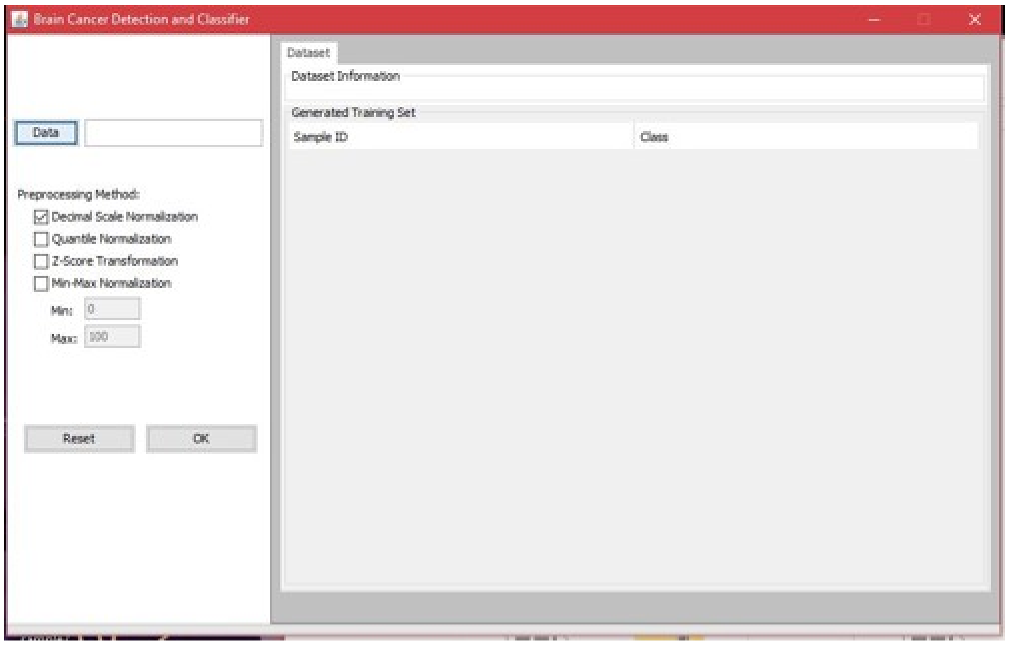
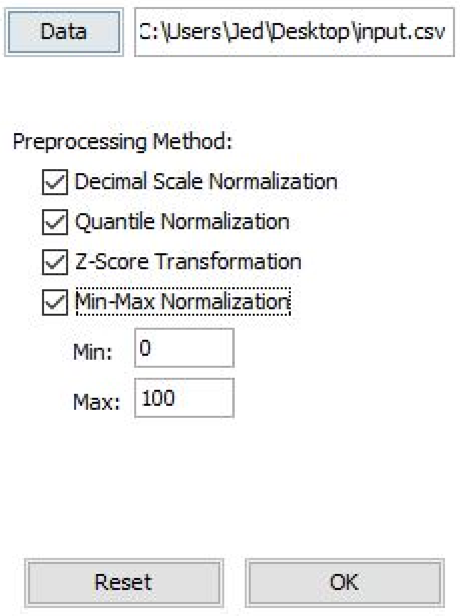


Figure 11. Main Menu

The side panel consists of the Data field where you can import your respective .csv file by pressing the Data button and the possible preproccessing methods to be used wherein the user can select more than one. Once the user has finished specifying its microarray data input and preprocessing methods, they can process to press the OK button to start the prediction and classification process as shown in Figure 12.



C:\Users\Jed\Desktop\input.csv

Preprocessing Method:

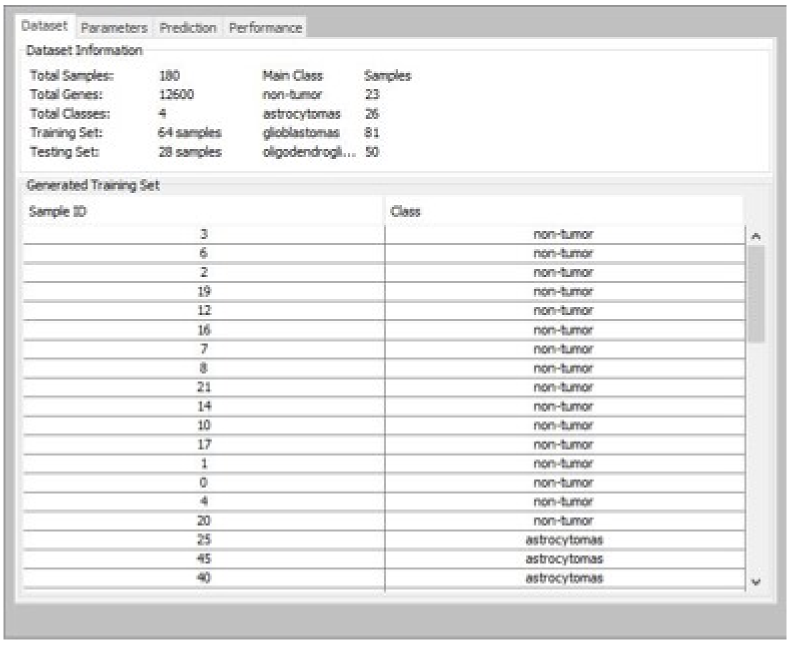
- [✓] Decimal Scale Normalization
- [✓] Quantile Normalization
- [✓] Z-Score Transformation
- [✓] Min-Max Normalization

Min: 0

Max: 100

Figure 12. Input

Once the classification process is finished, the generated results would be displayed in the results panel which first consists of the Dataset tab as shown wherein it views information regarding the csv input such as how many samples were placed, total amount of gene lists, number of glioma brain cancer types detected, etc.



Dataset Information

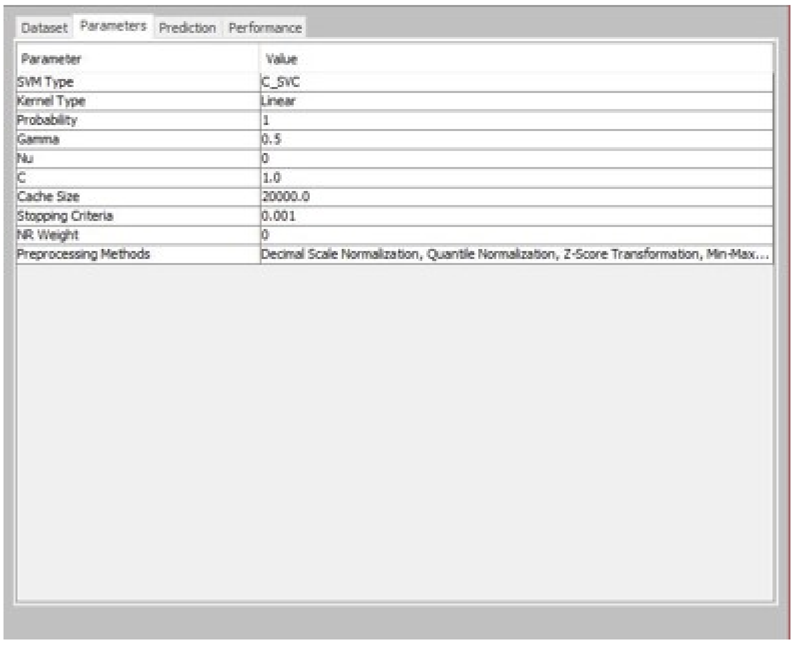
Total Samples:	180	Main Class	Samples
Total Genes:	12600	non-tumor	23
Total Classes:	4	astrocytomas	26
Training Set:	64 samples	glioblastomas	81
Testing Set:	28 samples	oligodendrogli...	50

Generated Training Set

Sample ID	Class
3	non-tumor
6	non-tumor
2	non-tumor
19	non-tumor
12	non-tumor
16	non-tumor
7	non-tumor
8	non-tumor
21	non-tumor
14	non-tumor
10	non-tumor
17	non-tumor
1	non-tumor
0	non-tumor
4	non-tumor
20	non-tumor
25	astrocytomas
45	astrocytomas
40	astrocytomas

Figure 13. Output, Dataset Tab

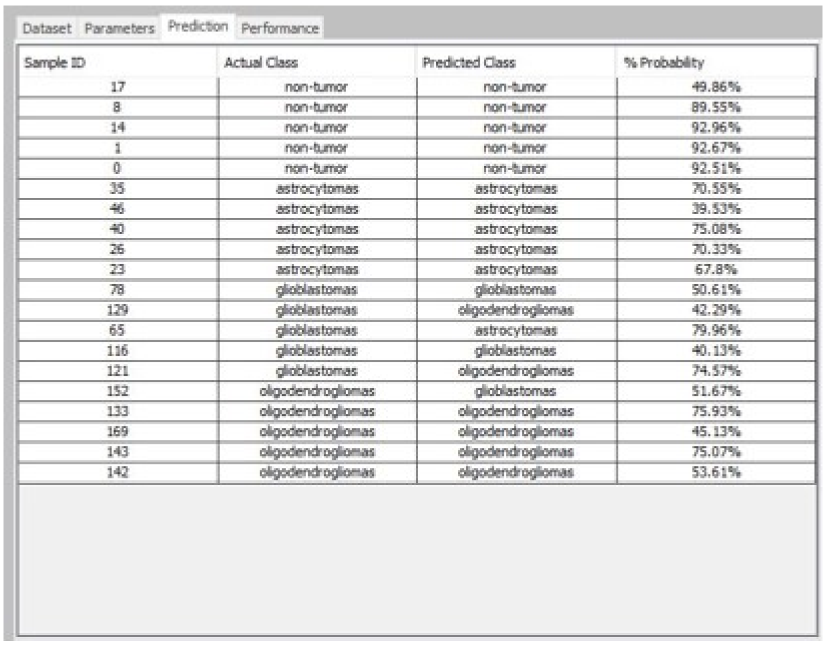
Next would be the Parameters tab which would consist of information the system accepted such as its SVM type, Preprocessing methods selected, etc.



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	Decimal Scale Normalization, Quantile Normalization, Z-Score Transformation, Min-Max...

Figure 14. Output, Parameters Tab

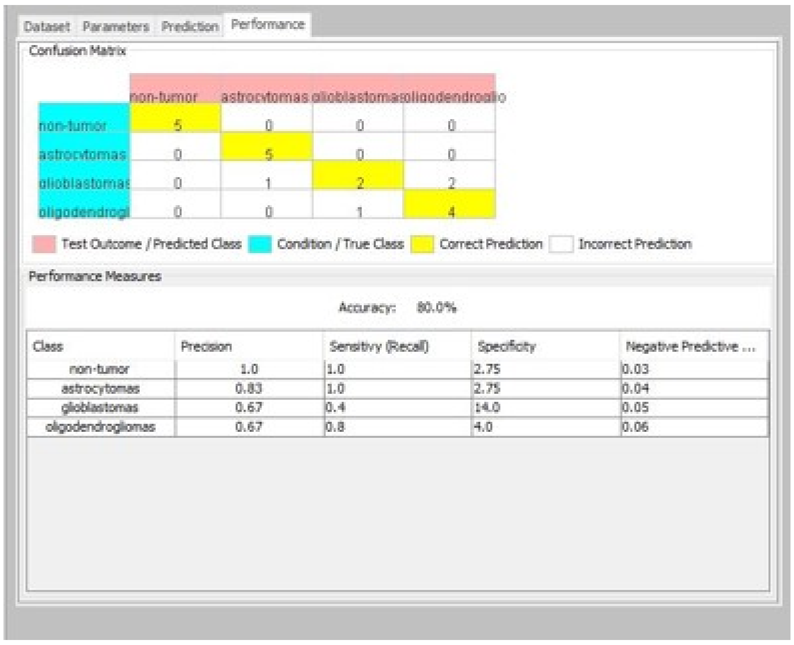
Next would be the Prediction tab that shows the prediction results of the testing set. It shows its sample ID, the predicted and actual class, and its % probability in prediction



Sample ID	Actual Class	Predicted Class	% Probability
17	non-tumor	non-tumor	49.86%
8	non-tumor	non-tumor	89.55%
14	non-tumor	non-tumor	92.96%
1	non-tumor	non-tumor	92.67%
0	non-tumor	non-tumor	92.51%
35	astrocytomas	astrocytomas	70.55%
46	astrocytomas	astrocytomas	39.53%
40	astrocytomas	astrocytomas	75.08%
26	astrocytomas	astrocytomas	70.33%
23	astrocytomas	astrocytomas	67.8%
78	glioblastomas	glioblastomas	50.61%
129	glioblastomas	oligodendrogliomas	42.29%
65	glioblastomas	astrocytomas	79.96%
116	glioblastomas	glioblastomas	40.13%
121	glioblastomas	oligodendrogliomas	74.57%
152	oligodendrogliomas	glioblastomas	51.67%
133	oligodendrogliomas	oligodendrogliomas	75.93%
169	oligodendrogliomas	oligodendrogliomas	45.13%
143	oligodendrogliomas	oligodendrogliomas	75.07%
142	oligodendrogliomas	oligodendrogliomas	53.61%

Figure 15. Output, Prediction Tab

Finally the Performance tab contains its evaluation measures on classifying the data using SVM by showing its Confusion Matrix, Overall accuracy, Precision, Sensitivity, Specificity, and Negative Predictive Value



	non-tumor	astrocytomas	glioblastomas	oligodendroglio
non-tumor	5	0	0	0
astrocytomas	0	5	0	0
glioblastomas	0	1	2	2
oligodendrogl	0	0	1	4

🟥 Test Outcome / Predicted Class 🟦 Condition / True Class 🟨 Correct Prediction ⬜ Incorrect Prediction

Accuracy: 80.0%

Class	Precision	Sensitivy (Recall)	Specificity	Negative Predictive ...
non-tumor	1.0	1.0	2.75	0.03
astrocytomas	0.83	1.0	2.75	0.04
glioblastomas	0.67	0.4	14.0	0.05
oligodendrogliomas	0.67	0.8	4.0	0.06

Figure 16. Output, Performance Tab

VI. Discussions

The dataset contains a total of 180 samples, 12600 gene expression values, and 4 Glioma Brain Cancer classes:

- 23 samples from epilepsy patients (non-tumor)
- 26 astrocytoma tumor samples
- 50 oligodendroglioma tumor samples
- 81 glioblastoma tumor samples

Using the given dataset it would then randomly generate the training set and testing set. Random generation without replacement would ensure that biases on a given glioma brain tumor type would be avoided in generating its training and testing sets. The system randomly selects 70% of the samples to be used as its training set and the remaining 30% as its testing set. Undersampling is used to address inequalities in the sample amount.

Ten-fold cross validation is performed on the dataset for each combination of the different preprocessing methods. The following table shows the results:

Preprocessing Methods	Accuracy
1	46.686
2	69.046
3	69.682
4	71.007
1, 2	66.663
1, 3	70.943
1, 4	71.642
2, 3	74.664
2, 4	74.74
3, 4	74.679
1, 2, 3	73.885
1, 2, 4	75.645
1, 3, 4	78.047
2, 3, 4	76.62
1, 2, 3, 4	76.352

1 - Decimal Scale Normalization, 2 - Quantile Normalization

3 - Z-score Transformation, 4 - Min-max Normalization

Figure 17. Ten-Fold Cross-validation results

The cross-validation results shows that the output's accuracy has improved as more preprocessing methods were added in combination. This implies that the prediction relies on how the data was preprocessed.

The results have shown that Min-max Normalization yielded the highest accuracy when used as the only preprocessing method, followed by Z-Score transformation. Decimal scale normalization yielded a very low accuracy overall because the method involved simple linear rescaling of the data into a decimal scale.

While most of the preprocessing methods were not very effective on their own, when combined together with other methods its performance improves by a larger margin with the combination of Decimal Scale Normalization, Z-score transformation, and Min-max nomalization boasting the highest accuracy (78.047%).

VII. Conclusions

This classification tool is a decision-support tool that implements Support Vector Machines to predict the type of Glioma Brain Cancer type that exists within gene expression samples of glioma brain tumors. It provides preprocessing methods to remove biases and external factors that might affect the prediction. It then presents the results in tables and provides additional information such as its accuracy and probability.

By adding more preprocessing methods, the reduction of genes would aid in giving higher accuracy results. Although this might be able to conduct prediction and analysis, limitations on both the existing hardware and limited amount of input would imply that this would have more rooms of improvement to generate more consistent and accurate results.

VIII. Recommendations

The system might be able to preprocess microarray data but that does not mean that these are the only preprocessing methods that can be implemented. One of the improvements is having additional preprocessing methods that would be also be able to remove non-biological factors and noise in gene expression data and be able to improve its specificity and accuracy, which are things that microarray data have low values of.

Another improvement is to have additional microarray samples for the few samples that have been placed in training and testing sets have affected the wide range of its accuracy. Having a few samples used for testing data meant that even a single false prediction would have a great impact in its overall accuracy so having more samples would make the evaluation of predicted results more consistent.

Lastly, this study could be improved if you could determine genes that would have a possible factor in possessing a particular condition, which in this case, the type of glioma brain cancer present. Being able to identify the genes that have consistently been positive for a particular glioma brain cancer type, and using them to evaluate other samples would make the process of diagnosing faster and more accurate.

IX. Bibliography

[1] N. C. Institute, “Brain cancerpatient version.”

[2] C. P. Davis, “Brain cancer.”

[3] N. C. Institute, “Seer stat fact sheets: Brain and other nervous system cancer.”

[4] S. Farrimond, *Report of the Second World Summit for Brain Tumour Patient Advocates.*

[5] N. B. T. Society, “Tumor types: Understanding brain tumors.”

[6] Cancer.net, “Brain tumor: Diagnosis.”

[7] R. Shai, T. Shi, T. J. Kremen, S. Horvath, L. M. Liau, T. F. Cloughesy, P. S. Mischel, and S. F. Nelson, “Gene expression profiling identifies molecular subtypes of gliomas,” *Oncogene*, vol. 22, no. 31, pp. 4918–4923, 2003.

[8] R. Jaksik, M. Iwanaszko, J. Rzeszowska-Wolny, and M. Kimmel, “Microarray experiments and factors which affect their reliability,” *Biology direct*, vol. 10, no. 1, p. 46, 2015.

[9] Y. Leung, C. Chang, Y. Hung, and P. Fung, “Gene selection in microarray data analysis for brain cancer classification,” in *2006 IEEE International Workshop on Genomic Signal Processing and Statistics*, pp. 99–100, IEEE, 2006.

[10] W. A. Freije, F. E. Castro-Vargas, Z. Fang, S. Horvath, T. Cloughesy, L. M. Liau, P. S. Mischel, and S. F. Nelson, “Gene expression profiling of gliomas strongly predicts survival,” *Cancer research*, vol. 64, no. 18, pp. 6503–6510, 2004.

[11] C. L. Nutt, D. Mani, R. A. Betensky, P. Tamayo, J. G. Cairncross, C. Ladd, U. Pohl, C. Hartmann, M. E. McLaughlin, T. T. Batchelor, *et al.*, “Gene

expression-based classification of malignant gliomas correlates better with survival than histological classification," *Cancer research*, vol. 63, no. 7, pp. 1602–1607, 2003.

[12] K. J. M. Bhavana Ghotekar, "Brain tumor detection and classification using svm,"

[13] S. Bauer, L.-P. Nolte, and M. Reyes, "Fully automatic segmentation of brain tumor images using support vector machine classification in combination with hierarchical conditional random field regularization," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 354–361, Springer, 2011.

[14] G. George and V. C. Raj, "Review on feature selection techniques and the impact of svm for cancer classification using gene expression profile," *arXiv preprint arXiv:1109.1062*, 2011.

[15] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 389–422, 2002.

[16] T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data," *Bioinformatics*, vol. 16, no. 10, pp. 906–914, 2000.

[17] S. Ramaswamy, P. Tamayo, R. Rifkin, S. Mukherjee, C.-H. Yeang, M. Angelo, C. Ladd, M. Reich, E. Latulippe, J. P. Mesirov, *et al.*, "Multiclass cancer diagnosis using tumor gene expression signatures," *Proceedings of the National Academy of Sciences*, vol. 98, no. 26, pp. 15149–15154, 2001.

[18] Y. Lee and C.-K. Lee, "Classification of multiple cancer types by multicategory support vector machines using gene expression data," *Bioinformatics*, vol. 19, no. 9, pp. 1132–1139, 2003.

[19] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques.* Elsevier, 2011.

[20] C. Cheadle, M. P. Vawter, W. J. Freed, and K. G. Becker, "Analysis of microarray data using z score transformation," *The Journal of molecular diagnostics*, vol. 5, no. 2, pp. 73–81, 2003.

[21] B. M. Bolstad, R. A. Irizarry, M. Åstrand, and T. P. Speed, "A comparison of normalization methods for high density oligonucleotide array data based on variance and bias," *Bioinformatics*, vol. 19, no. 2, pp. 185–193, 2003.

[22] A. Mandal, "What are genes?."

[23] H. H. M. Institute, "How to analyze dna microarray data."

[24] G. H. Reference, "What is dna?."

[25] K. Strimbu and J. A. Tavel, "What are biomarkers?," *Current Opinion in HIV and AIDS*, vol. 5, no. 6, p. 463, 2010.

[26] N. C. for Biotechnology Information, "Microarrays: Chipping away at the mysteries of science and medicine."

[27] H. H. M. Institute, "How to analyze dna microarray data."

[28] M. Rouse, "What is machine learning?."

[29] J. Brownlee, "Supervised and unsupervised learning algorithms."

[30] M. Kim, *Statistical Classification.* 2010.

[31] S. Reddy, "Machine learning: Classification."

[32] M. Mandel, "Lecture 3: Machine learning, classification, and generative models," *Machine learning*, vol. 1, p. 43, 2008.

[33] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," 2007.

[34] R. Schapire, "Machine learning algorithms for classification," *Princeton University*, vol. 10, 2006.

[35] OpenCV, "Introduction to support vector machines."

[36] K. S. Durgesh and B. Lekha, "Data classification using support vector machine," *Journal of Theoretical and Applied Information Technology*, vol. 12, no. 1, pp. 1–7, 2010.

[37] S. Mukherjee, "Classifying microarray data using support vector machines," in *A practical approach to microarray data analysis*, pp. 166–185, Springer, 2003.

[38] F. Markowetz, "Classification by support vector machines," *Practical DNA Microarray Analysis*, 2003.

[39] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, "A practical guide to support vector classification," 2003.

[40] "12.6 machine learning using svm."

[41] M. P. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, M. Ares, and D. Haussler, "Knowledge-based analysis of microarray gene expression data by using support vector machines," *Proceedings of the National Academy of Sciences*, vol. 97, no. 1, pp. 262–267, 2000.

[42] D. Anguita, S. Ridella, and D. Sterpi, "Testing the augmented binary multiclass svm on microarray data," in *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pp. 1966–1968, IEEE, 2006.

[43] J. J. Dai, L. Lieu, and D. Rocke, "Dimension reduction for classification with gene expression microarray data," *Statistical applications in genetics and molecular biology*, vol. 5, no. 1, p. 1147, 2006.

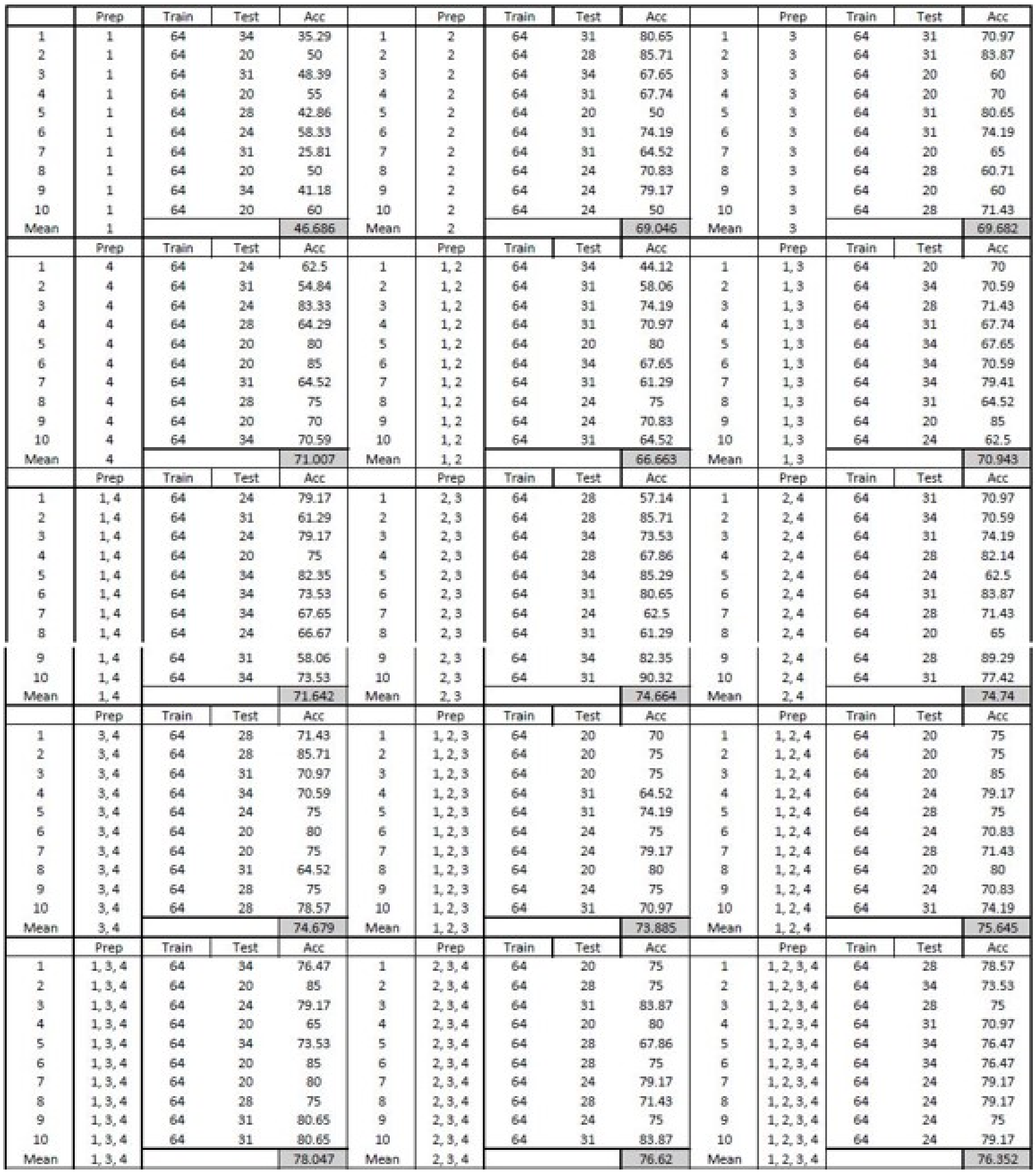
[44] A. Karatzoglou, D. Meyer, and K. Hornik, "Support vector machines in r," 2005.

[45] A. Bhattacharjee, W. G. Richards, J. Staunton, C. Li, S. Monti, P. Vasa, C. Ladd, J. Beheshti, R. Bueno, M. Gillette, *et al.*, "Classification of human lung carcinomas by mrna expression profiling reveals distinct adenocarcinoma subclasses," *Proceedings of the National Academy of Sciences*, vol. 98, no. 24, pp. 13790–13795, 2001.

X. Appendix

A. Forms

Ten-fold Cross Validation Results



	Prep	Train	Test	Acc		Prep	Train	Test	Acc		Prep	Train	Test	Acc
1	1	64	34	35.29	1	2	64	31	80.65	1	3	64	31	70.97
2	1	64	20	50	2	2	64	28	85.71	2	3	64	31	83.87
3	1	64	31	48.39	3	2	64	34	67.65	3	3	64	20	60
4	1	64	20	55	4	2	64	31	67.74	4	3	64	20	70
5	1	64	28	42.86	5	2	64	20	50	5	3	64	31	80.65
6	1	64	24	58.33	6	2	64	31	74.19	6	3	64	31	74.19
7	1	64	31	25.81	7	2	64	31	64.52	7	3	64	20	65
8	1	64	20	50	8	2	64	24	70.83	8	3	64	28	60.71
9	1	64	34	41.18	9	2	64	24	79.17	9	3	64	20	60
10	1	64	20	60	10	2	64	24	50	10	3	64	28	71.43
Mean	1			46.686	Mean	2			69.046	Mean	3			69.682
	Prep	Train	Test	Acc		Prep	Train	Test	Acc		Prep	Train	Test	Acc
1	4	64	24	62.5	1	1, 2	64	34	44.12	1	1, 3	64	20	70
2	4	64	31	54.84	2	1, 2	64	31	58.06	2	1, 3	64	34	70.59
3	4	64	24	83.33	3	1, 2	64	31	74.19	3	1, 3	64	28	71.43
4	4	64	28	64.29	4	1, 2	64	31	70.97	4	1, 3	64	31	67.74
5	4	64	20	80	5	1, 2	64	20	80	5	1, 3	64	34	67.65
6	4	64	20	85	6	1, 2	64	34	67.65	6	1, 3	64	34	70.59
7	4	64	31	64.52	7	1, 2	64	31	61.29	7	1, 3	64	34	79.41
8	4	64	28	75	8	1, 2	64	24	75	8	1, 3	64	31	64.52
9	4	64	20	70	9	1, 2	64	24	70.83	9	1, 3	64	20	85
10	4	64	34	70.59	10	1, 2	64	31	64.52	10	1, 3	64	24	62.5
Mean	4			71.007	Mean	1, 2			66.663	Mean	1, 3			70.943
	Prep	Train	Test	Acc		Prep	Train	Test	Acc		Prep	Train	Test	Acc
1	1, 4	64	24	79.17	1	2, 3	64	28	57.14	1	2, 4	64	31	70.97
2	1, 4	64	31	61.29	2	2, 3	64	28	85.71	2	2, 4	64	34	70.59
3	1, 4	64	24	79.17	3	2, 3	64	34	73.53	3	2, 4	64	31	74.19
4	1, 4	64	20	75	4	2, 3	64	28	67.86	4	2, 4	64	28	82.14
5	1, 4	64	34	82.35	5	2, 3	64	34	85.29	5	2, 4	64	24	62.5
6	1, 4	64	34	73.53	6	2, 3	64	31	80.65	6	2, 4	64	31	83.87
7	1, 4	64	34	67.65	7	2, 3	64	24	62.5	7	2, 4	64	28	71.43
8	1, 4	64	24	66.67	8	2, 3	64	31	61.29	8	2, 4	64	20	65
9	1, 4	64	31	58.06	9	2, 3	64	34	82.35	9	2, 4	64	28	89.29
10	1, 4	64	34	73.53	10	2, 3	64	31	90.32	10	2, 4	64	31	77.42
Mean	1, 4			71.642	Mean	2, 3			74.664	Mean	2, 4			74.74
	Prep	Train	Test	Acc		Prep	Train	Test	Acc		Prep	Train	Test	Acc
1	3, 4	64	28	71.43	1	1, 2, 3	64	20	70	1	1, 2, 4	64	20	75
2	3, 4	64	28	85.71	2	1, 2, 3	64	20	75	2	1, 2, 4	64	20	75
3	3, 4	64	31	70.97	3	1, 2, 3	64	20	75	3	1, 2, 4	64	20	85
4	3, 4	64	34	70.59	4	1, 2, 3	64	31	64.52	4	1, 2, 4	64	24	79.17
5	3, 4	64	24	75	5	1, 2, 3	64	31	74.19	5	1, 2, 4	64	28	75
6	3, 4	64	20	80	6	1, 2, 3	64	24	75	6	1, 2, 4	64	24	70.83
7	3, 4	64	20	75	7	1, 2, 3	64	24	79.17	7	1, 2, 4	64	28	71.43
8	3, 4	64	31	64.52	8	1, 2, 3	64	20	80	8	1, 2, 4	64	20	80
9	3, 4	64	28	75	9	1, 2, 3	64	24	75	9	1, 2, 4	64	24	70.83
10	3, 4	64	28	78.57	10	1, 2, 3	64	31	70.97	10	1, 2, 4	64	31	74.19
Mean	3, 4			74.679	Mean	1, 2, 3			73.885	Mean	1, 2, 4			75.645
	Prep	Train	Test	Acc		Prep	Train	Test	Acc		Prep	Train	Test	Acc
1	1, 3, 4	64	34	76.47	1	2, 3, 4	64	20	75	1	1, 2, 3, 4	64	28	78.57
2	1, 3, 4	64	20	85	2	2, 3, 4	64	28	75	2	1, 2, 3, 4	64	34	73.53
3	1, 3, 4	64	24	79.17	3	2, 3, 4	64	31	83.87	3	1, 2, 3, 4	64	28	75
4	1, 3, 4	64	20	65	4	2, 3, 4	64	20	80	4	1, 2, 3, 4	64	31	70.97
5	1, 3, 4	64	34	73.53	5	2, 3, 4	64	28	67.86	5	1, 2, 3, 4	64	34	76.47
6	1, 3, 4	64	20	85	6	2, 3, 4	64	28	75	6	1, 2, 3, 4	64	34	76.47
7	1, 3, 4	64	20	80	7	2, 3, 4	64	24	79.17	7	1, 2, 3, 4	64	24	79.17
8	1, 3, 4	64	28	75	8	2, 3, 4	64	28	71.43	8	1, 2, 3, 4	64	24	79.17
9	1, 3, 4	64	31	80.65	9	2, 3, 4	64	24	75	9	1, 2, 3, 4	64	24	75
10	1, 3, 4	64	31	80.65	10	2, 3, 4	64	31	83.87	10	1, 2, 3, 4	64	24	79.17
Mean	1, 3, 4			78.047	Mean	2, 3, 4			76.62	Mean	1, 2, 3, 4			76.352

B. Source Code

1. main/Main.java

```
package main;

import user_interface.UserInterface;

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

2. input/Input.java

```
package input;

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 Input
{
        private JFrame appFrame;
        private String file;
        private int totalFeatures, totalSamples, totalClasses;
        private int totalTrainSamples, totalTestSamples;
        private int noOfSamplesPerClassForTesting, noOfSamplesPerClassForTraining;
        private String[] probeSet;
        private double[][] trainingSet, testingSet, 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 svm_model svmModel;
        private boolean error = false;

        public Input (String file, boolean subclassify, JFrame appFrame)
        {
                this.appFrame = appFrame;
                this.file = file;
                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 int getNoOfSamplesPerClassForTraining () { return noOfSamplesPerClassForTraining; }
        public svm_model getSvmModel () { return svmModel; }

        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 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++)
        {
            int dataSizePerClass = dataPerClass.get(i).size();
            int numberOfTrainingSamples = noOfSamplesPerClassForTraining;
            int diff = dataSizePerClass - numberOfTrainingSamples;

            if (diff < 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 +
                        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).siz

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

                        double[] sample = new double[fullDataset.get(0).length +
                        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 (noOfSamplesPerClassForTrai
                trainingSamplesIndex.add(chosenSamples);

                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(fullDataset.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();

        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;
    }
}
```

3. preprocesses/DecimalScaleNormalization.java

```
package preprocesses;

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

    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
            }
        }
        return preprocessedData;
    }

    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;
    }
}
```

4. preprocesses/QuantileNormalization.java

```
/** Quantile Normalization:
* 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 preprocesses;

import java.util.ArrayList;

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

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

        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);
        }

        ArrayList <ArrayList <double[]>> transpose = transposeData(data, totalSamples, totalFeatu

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

        data = untransposeData(transpose, totalSamples, totalFeatures);

        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) * 10000) / (double)10000;
        }

        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);
            }
        }

        processedData = unsortMatrix (data, totalSamples, totalFeatures);
        return processedData;
    }

    private static ArrayList <ArrayList <double[]>> transposeData (ArrayList <ArrayList <double[]>> d
    {
        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;
    }

    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;
    }

    private static ArrayList <ArrayList <double[]>> untransposeData (ArrayList <ArrayList <double[]>>
    {
        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 double[][] unsortMatrix(ArrayList<ArrayList<double[]>> data, int totalSamples, int
    {
        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;
        }
}
```

5. preprocesses/ZScoreTransformation.java

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

package preprocesses;

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

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

                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] - meanIntensitie
                        }
                }
                return processedData;
        }

        private static double[] getMeanIntensityPerFeature (double[][] dataset, int totalSamples, int tot
        {
                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++)
                        {
                                means[col] += (double) dataset[row][col];
                        }
                }

                for (int i=1; i<totalFeatures; i++)
                {
                        means[i] = (double)means[i] / (double)totalSamples;
                }

                return means;
        }

        //FORMULA: sqrt(sum((x-mean(x))^2)/(n-1))
        private static double[] getStdDeviationPerFeature (double[] meanPerFeature, double[][] dataset, i
        {
                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++)
                        {
                                stdDev[col] += (double) (dataset[row][col] - meanPerFeature[col]) * (dou
                        }
                }

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

                return stdDev;
        }
}
```

6. preprocesses/MinMaxNormalization.java

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

```
*/
package preprocesses;

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

        public static double[][] performMethod(double[][] dataset, int totalSamples, int totalFeatures, i
        {
                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]) /
                        }
                }

                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;
        }
}
```

7. process/ProcessInput.java

```
package process;

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

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

import input.Input;
import libsvm.svm_parameter;
import getset.GettersSetters;

import preprocesses.DecimalScaleNormalization;
import preprocesses.MinMaxNormalization;
import preprocesses.QuantileNormalization;
import preprocesses.ZScoreTransformation;

import svm.SvmEvaluator;
import svm.SvmParameters;
import svm.SvmTrainer;

import user_interface.ResultDatasetTab;
import user_interface.ResultParametersTab;
import user_interface.ResultPredictionTab;
import user_interface.ResultStatsTab;

public class ProcessInput extends SwingWorker <JTabbedPane, Void>
{
        private final int PREPROCESSING_METHODS = 4;
        private JTabbedPane resultTabbedPane;
        private JFrame appFrame;
        private Input dataModel;
        private GettersSetters settings;
        private svm_parameter svmParams;
        private ArrayList <JTabbedPane> resultTabbedPanes;

        public ProcessInput (Input dataModel, GettersSetters settings, JTabbedPane resultTabbedPane, JFra
        {
                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_SCROL
                resultTabbedPane.addTab (”Prediction”, new JScrollPane (null, JScrollPane.VERTICAL_SCROL
                resultTabbedPane.addTab(”Performance”, new JScrollPane (null, JScrollPane.VERTICAL_SCROL

                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(1, new JScrollPane (new ResultParametersTab (svmParams, s
                resultTabbedPane.setEnabledAt(1, true);
                resultTabbedPane.setSelectedIndex(1);
                setProgress (30);

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

                        if ((!settings.getPrep()[0] && !settings.getPrep()[1] && !settings.getPrep()[2] &
                        {
                                trainingSet = DecimalScaleNormalization.preprocessData(trainingSet, dataM
                                testingSet = DecimalScaleNormalization.preprocessData(testingSet, dataMo
                        }
                        if (settings.getPrep()[1])
                        {
                                trainingSet = QuantileNormalization.preprocessData(trainingSet, dataModel
                                testingSet = QuantileNormalization.preprocessData(testingSet, dataModel.g
                        }

                        if (settings.getPrep()[2])
                        {
                                trainingSet = ZScoreTransformation.preprocessData(trainingSet, dataModel.
                                testingSet = ZScoreTransformation.preprocessData(testingSet, dataModel.ge
                        }

                        if (settings.getPrep()[3])
                        {
                                trainingSet = MinMaxNormalization.preprocessData(trainingSet, dataModel.g
                                testingSet = MinMaxNormalization.preprocessData(testingSet, dataModel.get
                        }

                        dataModel.setTrainingSet(trainingSet);
                        dataModel.setTestingSet(testingSet);
                        setProgress (60);

                        trainingSet = dataModel.getTrainingSet();
                        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!", "W
                                }
                                else
                                {
                                        trainingResults[j] = SvmEvaluator.evaluate(dataModel.getTestingSe
                                }

                                dataModel.setTrainingResults (trainingResults);
                        }
                }

                resultTabbedPane.setComponentAt(2, new JScrollPane (new ResultPredictionTab (dataModel),
                resultTabbedPane.setEnabledAt(2, true);
                resultTabbedPane.setSelectedIndex(2);
                setProgress (90);

                resultTabbedPane.setComponentAt(3, new JScrollPane (new ResultStatsTab (dataModel), JScro
                resultTabbedPane.setEnabledAt(3, true);
                resultTabbedPane.setSelectedIndex(3);
                setProgress(100);

                return resultTabbedPane;
        }

        protected void done ()
        {
                try
                {
                        JTabbedPane resultPane = get ();
                        resultTabbedPanes.add(resultPane);
                }
                catch (InterruptedException e)
                {
                        JOptionPane.showMessageDialog(appFrame, "Process interrupted!", "Warning", JOptio
                        e.printStackTrace();
                        return;
                }
                catch (ExecutionException e)
                {
                        JOptionPane.showMessageDialog(appFrame, "Execution Exception!", "Warning", JOptio
                        e.printStackTrace();
                        return;
                }
                catch (CancellationException e)
                {
                        JOptionPane.showMessageDialog(appFrame, "Process cancelled!", "Warning", JOptionP
                        e.printStackTrace();
                        return;
                }
                catch (NullPointerException e)
                {
                        JOptionPane.showMessageDialog(appFrame, "Process Completed!", "Information", JOp
                        //      e.printStackTrace();
                        return;
                }
        }
}
```

8. getset/GettersSetters.java

```
package getset;

public class GettersSetters
{
        private boolean decnorm = true;
        private boolean qnorm = false;
        private boolean zscore = false;
        private boolean minmax = false;
        private int min = 0, max = 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 setMinMaxInterval (int min, int max)
        {
                this.min = min;
                this.max = max;
        }

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

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

9. 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)
        {
                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;

                for (int i=0; i<totalClasses; i++)
                {
                        result[i+2] = prob_estimates[i];
                }

                return result;
        }
}
```

10. 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];

                // 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;
        }
}
```

11. svm/SvmTrainer.java

```
package svm;

import input.Input;
import libsvm.*;

public class SvmTrainer
{
        public static svm_model getSvmModel(svm_parameter param, Input dataset)
        {
```

```
		svm_model model = trainSystem (param, dataset.getTrainingSet(), dataset.getTotalTrainSam
		return model;
	}

	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);
	}
}
```

12. 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 tmp=h.len; h.len=len; len=tmp;} 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[] tmp=head[i].data; head[i].data=head[j].data; head[j].data=tmp;} while(false);
                do {int tmp=head[i].len; head[i].len=head[j].len; head[j].len=tmp;} 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 tmp=i; i=j; j=tmp;} 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 tmp=h.data[i]; h.data[i]=h.data[j]; h.data[j]=tmp;} wh
                                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[] tmp=x[i]; x[i]=x[j]; x[j]=tmp;} while(false);
                if(x_square != null) do {double tmp=x_square[i]; x_square[i]=x_square[j]; x_square[j]=tmp
        }

        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 tmp=y[i]; y[i]=y[j]; y[j]=tmp;} while(false);
                do {double tmp=G[i]; G[i]=G[j]; G[j]=tmp;} while(false);
                do {byte tmp=alpha_status[i]; alpha_status[i]=alpha_status[j]; alpha_status[j]=tmp;} whi
                do {double tmp=alpha[i]; alpha[i]=alpha[j]; alpha[j]=tmp;} while(false);
                do {double tmp=p[i]; p[i]=p[j]; p[j]=tmp;} while(false);
```

```
		do {int tmp=active_set[i]; active_set[i]=active_set[j]; active_set[j]=tmp;} while(false);
		do {double tmp=G_bar[i]; G_bar[i]=G_bar[j]; G_bar[j]=tmp;} 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 || Gmin_idx == -1)
                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 || Gmin_idx == -1)
                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 tmp=y[i]; y[i]=y[j]; y[j]=tmp;} while(false);
		do {double tmp=QD[i]; QD[i]=QD[j]; QD[j]=tmp;} 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 tmp=QD[i]; QD[i]=QD[j]; QD[j]=tmp;} 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 tmp=sign[i]; sign[i]=sign[j]; sign[j]=tmp;} while(false);
                do {int tmp=index[i]; index[i]=index[j]; index[j]=tmp;} while(false);
                do {double tmp=QD[i]; QD[i]=QD[j]; QD[j]=tmp;} 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=322;
        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,
        {
```

```
		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 tmp=perm[i]; perm[i]=perm[j]; perm[j]=tmp;} 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 dist
        return mae;
}

// label: label name, start: begin of each class, count: #data of classes, perm: indices to the o
// 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, in
{
        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 correspondin
        //
        if (nr_class == 2 && label[0] == -1 && label[1] == +1)
        {
                do {int tmp=label[0]; label[0]=label[1]; label[1]=tmp;} while(false);
                do {int tmp=count[0]; count[0]=count[1]; count[1]=tmp;} 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 detail

                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]+"
                        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],
                        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, doub
        {
                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 tmp=index[start[c]+j]; index[start[c]+j]=index[start[c]+i
                                }
                        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 tmp=perm[i]; perm[i]=perm[j]; perm[j]=tmp;} 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]
			}
			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_pa
		    model.probA!=null)
		return model.probA[0];
		else
		{
			System.err.print("Model doesn't contain information for SVR probability inference
			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_estimat
{
        if ((model.param.svm_type == svm_parameter.C_SVC || model.param.svm_type == svm_parameter
            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[
                                pairwise_prob[j][i]=1-pairwise_prob[i][j];
                                k++;
                        }
                if (nr_class == 2)
                {
                        prob_estimates[0] = pairwise_prob[0][1];
                        prob_estimates[1] = pairwise_prob[1][0];
                }
                else
```

```
                        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(

        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);
}

private static boolean read_model_header(BufferedReader fp, svm_model model)
{
        svm_parameter param = new svm_parameter();
        model.param = param;
        // parameters for training only won't be assigned, but arrays are assigned as NULL for sa
        param.nr_weight = 0;
        param.weight_label = null;
        param.weight = null;

        try
        {
                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 false;
                                }
                        }
                        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 false;
                                }
                        }
                        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 false;
                                }
                        }
                }
                catch(Exception e)
                {
                        return false;
                }
                return true;
        }

        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();
                model.rho = null;
                model.probA = null;
                model.probB = null;
                model.label = null;
                model.nSV = null;

                if (read_model_header(fp, model) == false)
                {
                        System.err.print("ERROR: failed to read model\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_paramet
                model.probA!=null && model.probB!=null) ||
                ((model.param.svm_type == svm_parameter.EPSILON_SVR || model.param.svm_type == svm_param
                 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;
        }
}
```

13. 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]

        // 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
};
```

14. libsvm/svm_node.java

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

15. 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;
                }
        }

}
```

16. libsvm/svm_print_interface.java

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

17. libsvm/svm_problem.java

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

18. 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;
        }
}
```

19. user_interface/MatrixPanel.java

```
package user_interface;

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 ROW
                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;
                double h = height;
                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/
                                }
                                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
                                }
                        }
                }
        }
}
```

20. user_interface/ResultDatasetTab.java

```
package user_interface;

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 input.Input;

@SuppressWarnings("serial")
public class ResultDatasetTab extends JPanel {
        private Input 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 (Input 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"},
                };

                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(indexPerClass);
                                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;
        }
```

```
}
```

21. user_interface/ResultParametersTab.java

```
package user_interface;

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 getset.GettersSetters;

@SuppressWarnings("serial")
public class ResultParametersTab extends JPanel {
        public ResultParametersTab(svm_parameter p, GettersSetters 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]) || (!settings.getPrep()[0] && !settings.getPrep()[1] && !sett
                        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);

                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_NEEDE
                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}
        };

        @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;
        }
}
```

22. user_interface/ResultPredictionTab.java

```
package user_interface;

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 input.Input;

@SuppressWarnings("serial")
public class ResultPredictionTab extends JPanel {
        public ResultPredictionTab(Input 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
                                arr[2] = dataModel.getClassNames().get(predIndex);
                                arr[3] = Math.round((dataModel.getTrainingResults()[count][predIndex+2])
                                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

                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;
        }
}
```

23. user_interface/ResultStatsTab.java

```
package user_interface;

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 input.Input;

@SuppressWarnings("serial")
public class ResultStatsTab extends JPanel {
        private Input dataModel;
        private int[][] matrix;
        private double accuracy = 0;
        private double[] precision, sensitivity, specificity, negativePredictiveValue;

        public ResultStatsTab(Input 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 != 4) {
                        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
                        sensitivity[i] = Math.round(((double)truePositive[i] / (double)conditionPositive[
                        specificity[i] = Math.round(((double)trueNegative[i] / (double)conditionNegative[
                        negativePredictiveValue[i] = Math.round((double)trueNegative[i] / (double)testOut
                }
        }
}

@SuppressWarnings("serial")
class MeasuresTableModel extends AbstractTableModel {
        String[] columnNames = {"Class", "Precision", "Sensitivy (Recall)", "Specificity", "Negative Pre
        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;
        }
}
```

24. user_interface/SidePanel.java

```
package user_interface;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
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 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.JTabbedPane;
import javax.swing.JTextField;

import getset.GettersSetters;
import input.Input;
import process.ProcessInput;
import util.Utils;

@SuppressWarnings("serial")
public class SidePanel extends JPanel implements ActionListener, ItemListener {
        public int fileIndex = 1;
        private GridBagLayout gridbag = new GridBagLayout ();
        private GridBagConstraints c = new GridBagConstraints ();
        private JFrame appFrame;
        private JButton okBtn, openDatasetBtn;
        private JTextField fileTextField;
        private JLayeredPane newLPane;
        private JLayeredPane resultListPane;
        private JCheckBox decScaleCheckBox, qNormCheckBox, zScoreCheckBox, minMaxNormCheckBox;
        private JTextField minField, maxField;
        private JProgressBar progressBar;
        private JTabbedPane resultTabbedPane;
        private JButton startBtn, stopBtn;

        public SidePanel (JFrame appFrame, JTabbedPane resultTabbedPane, JProgressBar progressBar) {
                this.appFrame = appFrame;
                this.resultTabbedPane = resultTabbedPane;
                this.progressBar = progressBar;

                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")));

                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.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));

        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.setBackground(Color.WHITE);
        newLPane.setPreferredSize(new Dimension (237, 500));

        openDatasetBtn = new JButton ("Import .csv");
        openDatasetBtn.setBackground(Color.WHITE);
        openDatasetBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
        openDatasetBtn.setBounds(5, 75, 72, 25);
        openDatasetBtn.addActionListener(this);

        fileTextField = new JTextField ();
        fileTextField.setBounds(82, 75, 148, 25);

        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);

        okBtn = new JButton ("Submit");
        okBtn.addActionListener(this);
        okBtn.setBorder(BorderFactory.createLineBorder(Color.LIGHT_GRAY));
        okBtn.setBounds(125, 300, 100, 25);

        newLPane.add(openDatasetBtn);
        newLPane.add(fileTextField);
        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(okBtn);

                return newLPane;
        }

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

                if (src == openDatasetBtn) {
                        Utils.openFileChooser(fileTextField);
                }

                if (src == okBtn) {
                        if (fileTextField.getText().equals("")) {
                                JOptionPane.showMessageDialog(appFrame, "Choose an input dataset!", "War
                        }
                        else {
                                Input dataModel = new Input (fileTextField.getText(), false, appFrame);
                                if (!dataModel.hasError()) {
                                        fileIndex++;

                                        GettersSetters settings = new GettersSetters ();
                                        settings.setPrep(decScaleCheckBox.isSelected(), qNormCheckBox.isS

                                        if (minMaxNormCheckBox.isSelected()) {
                                                settings.setMinMaxInterval(Integer.parseInt(minField.getT
                                        }

                                        ProcessInput processor = new ProcessInput (dataModel, settings, r
                                        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.getStateChange() == ItemEvent.DESELECTED) {
                        if (e.getSource() == minMaxNormCheckBox) {
                                minField.setEnabled(false);
                                maxField.setEnabled(false);
                        }

                }
        }
}
```

25. user_interface/UserInterface.java

```
package user_interface;

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.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 GridBagLayout gridbag;
private GridBagConstraints c;

public UserInterface () {
        appFrame = new JFrame ("Glioma Brain Cancer Detection and Classifier");
        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 () {
        }

        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) {

        }
}
```

XI. Acknowledgement

OMAYGHAD MY PAMILEE

Una sa lahat gusto ko magpasalamat sa aking inay na walang-sawang kinulit ako upang tapusin ang SP ko. Sabi mo dahil sa kakakompyuter ko kaya ko nakamit ang pagtapos nito pero sa totoo lang dahil talaga sa patuloy mong pagpapaalala na gawin ko ang thesis ko kaya ako ginanahan ipagpatuloy ito. Maliban diyan, gusto ko rin magpasalamat sa'yo dahil sa kahit anong pagkukulang ko man sa panahon ko dito sa UP ay patuloy mo akong sinusuportahan at ginagawa ang buong makakaya upang maging masaya ako. We made it Ma, para sa'yo talaga 'tong pagtatapos ko!!

Sunod naman gusto ko magpasalamat sa mga propesor na gumabay sa'kin upang magawa ang SP na ito: si Sir Solano na unang nagbigay sa akin ng mga ideya paano papanindigan ang topic na 'to sa pamamagitan ng pagibibigay ng mga sources at ang kanyang suporta. Hindi man kayo ang naging adviser ko formally, ikaw pa rin ang unang nagbigay ng direksyon sa SP na 'to. Kay Ma'am Sheila na naging opisyal kong adviser maraming-maraming salamat din sa inyong paggabay sa aking SP lalo na sa mga tanong na ninais niyong paghandaan ko bago ang aking proposal at defense. Humihingi rin ako ng paumanhin kung hindi ako masyadong nagparamdam sa inyo ngunit maraming salamat at nagtiwala kayo sa akin at ipinaglaban ang schedules ng proposal at defense ko kahit naging biglaan lang ang consultations. Special mention rin kay Sir Berwin na mas malaki pa ang pag-aalala sa SP ko kaysa sa sarili ko. Maraming salamat at ipinaglaban mo pa na magdefend ako sa huling araw ng defense. Kung hindi dahil sa'yo baka hindi na ako umabot sa martsa ngayong taon. Xièxiè!

Sa mga alumni na tumulong rin sa akin upang magawa ko ito, sina Kuya Zach Marasigan para sa pagbigay ng sites kung saan makakakuha ng dataset para sa microarray data, at kay Ate Jen Cabrera na gumawa ng napakagandang thesis noon na nagbigay-inspirasyon sa SP na 'to. Maraming salamat talaga at malaki ang utang na loob ko sa inyo. Kung magkakaroon man ng panahon na mapapadaan ako sa inyong dalawa kung saan man, papakainin ko kayo promise.

Sunod naman sa blockmates ko. Maraming salamat at kayo ang bumuo ng kolehiyo ko. Special mention sa una kong tropa dito: Adi, Sheela, Lois, Moses. Sa forever kaklase kong si RS, at sa mga madalas ko ring nakasama: Reniel, Kyra, Joey, Riana, Reinier, Lei, Sheena, Francis. Salamat sa memorable na college experience!

Sa mga nakilala ko rin na hindi ko blockmates, maraming salamat sa suporta niyo sa akin. Special mention sa mga ComSci lower years, lalo na sa Batch 2014 at 2015 na napakasuportive (mang-asar lol), at sa mga kaibigan kong humihingi ng special mention: Master Hokage Reuben na laging maaasahan at karamay kong nag-super saiyan ng SP, at kay Buddy (Betina) na binigyan ako ng lakas at motivation sa kalagitnaan ng college life ko HAHA.

At gusto ko rin magpasalamat sa Panginoong Maykapal na binigyan ako ng pagkakataon upang makaraos sa lahat ng paghihirap na pinagdaanan ko. Salamat at binigyan Niyo po ako ng lakas ng loob harapin lahat ng hirap at problema na dumaan sa akin. Inaalay ko rin sa Inyo ang tagumpay na ito, more power!!