BENG 146 Project Description

Amin Zollanvari[a,*]

[a]*Department of Electrical and Electronic Engineering, Nazarbayev University, Astana, Kazakhstan*

Abstract

This is the description of the project given to students in Computer Programming for Engineers course at Nazarbeyev University, Fall 2016 (BENG 146). First we cover the background needed to execute the project. Then we follow by describing each project and the expected outcome of each project. The grading chart/style for evaluating projects will be described at the end of each project description.

Keywords: Classification, Pattern Recognition

1. Introduction

In the last 50 years, many researchers working in signal processing, statistics, and computer science communities have developed many techniques to extract information from different types of data. Most of "well-known" techniques in this avenue have been fashioned for situations in which a dataset of large sample size for few variables is available. However, nowadays in many practical situations we are facing datasets collected for a limited number of samples but very large number of variables–a scenario exactly opposite to what these techniques have been developed for. As such there is a pressing need to re-evaluate the performance of our "well-known" techniques in general, and in pattern recognition, in particular, and modify our techniques to suit our need.

Pattern recognition is the science of developing the set of mathematical, statistical and computational methods that automatize the process of finding specific patterns in a dataset. For this purpose, many mathematical models (classifiers, error estimators, feature selections,...) have been developed in the last few decades. Below we describe some well-known mathematical models in this field.

2. Classifiers

Throughout this document we represent column vectors by bold lower-case letters, the matrices by bold capital letters, and scalers just by regular lower-case letters.

Let us assume we have n independent observations, $S^n = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$ that have been independently collected from two populations Π_0 and Π_1. Here $\mathbf{x}_i$ is a column vector of p-dimension in which each dimension represents the value of one variable. In addition each y_i shows the *class* from which $\mathbf{x}_i$ is coming from. That is if y_i is 0, then it means that $\mathbf{x}_i$ has been taken from population Π_0 (or class 0) and if y_i is 1, then it means that $\mathbf{x}_i$ has been taken from population Π_1 (or class 1) . The goal in classification is to construct a mathematical model using *training data*, S^n to be able to classify a future sample point, $\mathbf{x}$ to the class that is coming from (either class 0 or 1). For this purpose we need to construct classifiers (for a practical example, refer to AnIntroductionToMachineLearning.pdf). In the following, we describe some of

*Corresponding author.
Email address: `amin.zollanvari@nu.edu.kz` (Amin Zollanvari)

classifiers in the field. In the following let S_j^n, $j \in 0, 1$ denote the subset of training sample that belong to class j, ($j = 0$ or 1). We denote the number of sample points coming from class 0 by n_0 and the number of sample points coming from class 1 by n_1 (so cardinality of S_0^n is n_0)

2.1. Linear Discriminant Analysis (LDA)

Linear discriminant function (LDF) [1] is defined by

$$W_{LDF}(\mathbf{x}) = \left(\mathbf{x} - \frac{\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1}{2}\right)^T \mathbf{C}^{-1}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1) = \mathbf{x}^T\mathbf{C}^{-1}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}) - \frac{\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1}{2}\mathbf{C}^{-1}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1) \tag{1}$$

where $\bar{\mathbf{x}}_0 = \frac{1}{n_0}\sum_{\mathbf{x}_l \in S_0^n} \mathbf{x}_l$ and $\bar{\mathbf{x}}_1 = \frac{1}{n_1}\sum_{\mathbf{x}_l \in S_1^n} \mathbf{x}_l$ being the sample means for each class and $\mathbf{C}$ being the pooled sample covariance matrix,

$$\mathbf{C} = \frac{(n_0 - 1)\mathbf{C}_0 + (n_1 - 1)\mathbf{C}_1}{n_0 + n_1 - 2} \tag{2}$$

where

$$\mathbf{C}_i = \frac{1}{n_i - 1}\sum_{\mathbf{x}_l \in S_i^n} (\mathbf{x}_l - \bar{\mathbf{x}}_i)(\mathbf{x}_l - \bar{\mathbf{x}}_i)^T \tag{3}$$

The LDA *classifier* is then given by

$$\psi_{LDA}(\mathbf{x}) = \begin{cases} 1\,, & \text{if } W_{LDF}(\mathbf{x}) \leq \mathrm{c} \\ 0\,, & \text{if } W_{LDF}(\mathbf{x}) > \mathrm{c} \end{cases}, \tag{4}$$

where $\mathrm{c} = \log\frac{(1-\alpha_0)}{\alpha_0}$ where α_0 is an estimate of class 0 prior probability. Therefore, in case there is a 50% chance that a random sample in the future belongs to class 0 or 1, then $\log\frac{(1-\alpha_0)}{\alpha_0} = 0$;

2.2. Diagonal Linear Discriminant Analysis (DLDA)

A closely related classifier to LDF classifier is the so-called diagonal LDA (DLDA). This classifier is defined using the DLDF defined by difference of this classifier with LDA is in the sample estimate of covariance matrix. To be more specific, DLDA is defined by

$$W_{DLDF}(\mathbf{x}) = \left(\mathbf{x} - \frac{\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1}{2}\right)^T \mathbf{D}^{-1}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)\,, \tag{5}$$

where $\mathbf{D}$ has the same diagonal elements as in $\mathbf{C}$ defined in (1) but with 0 off-diagonal elements. Similar to LDA, the classifier is then

$$\psi_{DLDA}(\mathbf{x}) = \begin{cases} 1\,, & \text{if } W_{DLDF}(\mathbf{x}) \leq \mathrm{c} \\ 0\,, & \text{if } W_{DLDF}(\mathbf{x}) > \mathrm{c} \end{cases}, \tag{6}$$

2.3. SVM

Linear-SVM finds the maximum-margin hyperplane that separates the set of points with $y_i = 1$ from $y_i = -1$ where y_i indicates the class labels, $i = 1, ..., n$. When the data is linearly separable, the normal column vector to the unique maximum-margin hyperplane, denoted by $\mathbf{v}$, is found by solving the following optimization problem

$$\begin{aligned} &\min_{\mathbf{v},c} ||\mathbf{v}|| \\ &\text{s.t.} \\ &y_i(\mathbf{v}^T\mathbf{x_i} - c) \geq 1 \end{aligned} \tag{7}$$

where $\mathbf{x_i}$ is a column vector data point, $i = 1, ..., n$. If the training data is not linearly separable, then this method can be modified by making appropriate changes to the optimization problem [2]. There is a type of SVM knows as radial basis function SVM (RBF SVM) in which the kernels that we use are different. More details can be found in [2].

2.4. G13 Classifier

This classifier is closely related to LDF and defined using the following discriminant function

$$W_{G_{13}}(\mathbf{x}) = \left(\left(\mathbf{x} - \frac{\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1}{2}\right)^T \mathbf{C}^{-1}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)\right)\left(\frac{n_0 + n_1 - 2 - p}{n_0 + n_1 - 2}\right), \tag{8}$$

Then the G13 classifier is defined as

$$\psi_{G13}(\mathbf{x}) = \begin{cases} 1\,, & \text{if } W_{G13}(\mathbf{x}) \leq \mathrm{c} \\ 0\,, & \text{if } W_{G13}(\mathbf{x}) > \mathrm{c} \end{cases}, \tag{9}$$

Note: In the C package that you work with, c term in (9) is in fact defined by $\log(\frac{n_1}{n_0})$ term (check the ldaTrn.c function in classifier.c). Therefore, to implement G13, you need to multiply only the LDA function part (not c) by $\frac{n_0+n_1-2-p}{n_0+n_1-2}$ where n_0 and n_1 are the number of sample points from class 0 and 1 and p is the dimension of vectors $\mathbf{x}$, $\bar{\mathbf{x}}_0$, $\bar{\mathbf{x}}_1$, which all are of the same dimension.

2.5. QDA Classifier

This classifier is defined using the following discriminant:

$$W_{QDF}(\mathbf{x}) = -\frac{1}{2}\left(\mathbf{x} - \bar{\mathbf{x}}_0\right)^T (\alpha\times\mathbf{I}_p+\mathbf{C}_0)^{-1}(\mathbf{x} - \bar{\mathbf{x}}_0)+\frac{1}{2}\left(\mathbf{x} - \bar{\mathbf{x}}_1\right)^T (\alpha\times\mathbf{I}_p+\mathbf{C}_1)^{-1}(\mathbf{x} - \bar{\mathbf{x}}_1)+\frac{1}{2}\log\left(\frac{|\alpha \times \mathbf{I}_p + \mathbf{C}_1|}{|\alpha \times \mathbf{I}_p + \mathbf{C}_0|}\right), \tag{10}$$

where α is a regularisation constant (choose some arbitrary number preferably larger than 0.1), $\mathbf{I}_p$ is a $p \times p$ identity matrix (diagonal matrix with 1's as diagonal elements) and $\bar{\mathbf{x}}_0 = \frac{1}{n_0}\sum_{\mathbf{x}_l \in S_0^n} \mathbf{x}_l$ and $\bar{\mathbf{x}}_1 = \frac{1}{n_1}\sum_{\mathbf{x}_l \in S_1^n} \mathbf{x}_l$ is the sample means for each class, $\mathbf{C}_i$ is the sample covariance matrix for each class as defined in (3), and $|\mathbf{C}_i|$ is the determinant of matrix $\mathbf{C}_i$.

$$\mathbf{C}_i = \frac{1}{n_i - 1}\sum_{\mathbf{x}_l \in S_i^n} (\mathbf{x}_l - \bar{\mathbf{x}}_i)(\mathbf{x}_l - \bar{\mathbf{x}}_i)^T \tag{11}$$

The QDA *classifier* is then given by

$$\psi_{QDA}(\mathbf{x}) = \begin{cases} 1\,, & \text{if } W_{QDF}(\mathbf{x}) \leq \mathrm{c} \\ 0\,, & \text{if } W_{QDF}(\mathbf{x}) > \mathrm{c} \end{cases}, \tag{12}$$

with an identical c as in (9). One can show that if $\mathbf{C}_0 = \mathbf{C}_1$, QDA and LDA are the same classifiers.

2.6. SEDC

Serdobolskii ([3]) has proposed a version of LDA classifier to which we refer as SEDC. To define SEDC, we define k dimensional vectors $\mathbf{y}$, $\bar{\mathbf{y}}_0$, and $\bar{\mathbf{y}}_1$, $(k < p)$, where each element in these vectors are obtained by taking average over every m consequtive variables of vectors $\mathbf{x}$, $\bar{\mathbf{x}}_0$, and $\bar{\mathbf{x}}_1$, respectively. To be able to do so, we assume k divides p; therefore, $p = km$. Some typical values of k are 2, 5, and 10 (ofcourse if p is divisible with k). Here is an example: let $p = 30$ and $m = 3$ (meaning $k = 10$). Therefore, vector $\mathbf{x}$ has dimension $p = 30$ and vector $\mathbf{y}$ has dimension $k = 10$. Then $\mathbf{y}$ is obtained by taking the average over every 3 consecutive variables in $\mathbf{x}$ as illustrated below:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{p-2} \\ x_{p-1} \\ x_p \end{bmatrix} \longrightarrow \mathbf{z} = \begin{bmatrix} \frac{x_1+x_2+x_3}{3} \\ \frac{x_4+x_5+x_6}{3} \\ \vdots \\ \frac{x_{p-2}+x_{p-1}+x_p}{3} \end{bmatrix} \tag{13}$$

In the same way we can define $\bar{\mathbf{z}}_0$, and $\bar{\mathbf{z}}_1$ out of $\bar{\mathbf{x}}_0$, and $\bar{\mathbf{x}}_1$. For example,

$$\bar{\mathbf{x}}_0 = \begin{bmatrix} \bar{x}_{0,1} \\ \bar{x}_{0,2} \\ \bar{x}_{0,3} \\ \bar{x}_{0,4} \\ \vdots \\ \bar{x}_{0,p-2} \\ \bar{x}_{0,p-1} \\ \bar{x}_{0,p} \end{bmatrix} \longrightarrow \bar{\mathbf{z}}_0 = \begin{bmatrix} \frac{\bar{x}_{0,1}+\bar{x}_{0,2}+\bar{x}_{0,3}}{3} \\ \frac{\bar{x}_{0,4}+\bar{x}_{0,5}+\bar{x}_{0,6}}{3} \\ \vdots \\ \frac{\bar{x}_{0,p-2}+\bar{x}_{0,p-1}+\bar{x}_{0,p}}{3} \end{bmatrix} \tag{14}$$

Then we can define SEDC using

$$W_{SEDC}(\mathbf{x}) = \left(\mathbf{z} - \frac{\bar{\mathbf{z}}_0 + \bar{\mathbf{z}}_1}{2}\right)^T (\bar{\mathbf{z}}_0 - \bar{\mathbf{z}}_1)\,. \tag{15}$$

Similarly, the classifier is then

$$\psi_{SEDC}(\mathbf{x}) = \begin{cases} 1\,, & \text{if } W_{SEDC}(\mathbf{x}) \leq \text{c} \\ 0\,, & \text{if } W_{SEDC}(\mathbf{x}) > \text{c} \end{cases}\,, \tag{16}$$

2.7. Regularised LDA, (RLDA)

This classifier is closely related to LDA and is defined using the following function:

$$W_{RLDF}(\mathbf{x}) = \left(\mathbf{x} - \frac{\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1}{2}\right)^T (\gamma \times \mathbf{I}_p + \mathbf{C})^{-1}(\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)\,, \tag{17}$$

and the RLDA *classifier* is then given by

$$\psi_{RLDA}(\mathbf{x}) = \begin{cases} 1\,, & \text{if } W_{RLDF}(\mathbf{x}) \leq \text{c} \\ 0\,, & \text{if } W_{RLDF}(\mathbf{x}) > \text{c} \end{cases}\,, \tag{18}$$

where $\mathbf{I}_p$ is the identity matrix and all parameters are similar to LDA (equations (2)-(3)) except for an additional γ parameter, which is a regularisation parameter similar to what we had in QDA but here we would like to find γ from the training data. To find the optimal value of γ, suppose you have a training data S^n of size n. In order to find optimal γ, we can randomly divide the data S^n to 2 folds of size $2n/3$ and $n/3$, namely $S_1^{2n/3}$ and $S_2^{n/3}$. Then:

Step 1: Choose a small positive initial value for γ, e.g., $\gamma_0 = 0.001$, and a prespecified large maximum value γ_{Max} , e.g., $\gamma_{Max} = 1000$ (it is up to you but the larger the better) to set a range for possible sets of γ, namely $R = [\gamma_0, \gamma_{Max}]$.

Step 2: Note that we would like to scan R to search for "best" γ but since R is continuos we have to discretize it first. Choose r values in the the range R (for example r could be 20 and you may choose a set of γ's that are spread with an exponential distance, or any other way you can think of) and index them as γ_i, $i = 1, .., r$. Set $i = 0$

Step 3: Set $i = i + 1$. Use $S_1^{2n/3}$ to train the classifier with γ_i. Find the "true error" of this classifier on $S_2^{n/3}$ (in the package see the functions ***Tst in calculateErrors.cpp function with *** currently being lda or svm). These functions calculate the true error but you need to properly feed them by $S_2^{n/3}$ here. Call this error ϵ_i. Record ϵ_i and its corresponding γ, i.e., γ_i.

Step 4: Is $i = r$? Is no, go to step 3. If yes, then the optimal γ is the γ that corresponds to minimum value among ϵ_1, ϵ_2, ..., ϵ_r. Call this γ, γ_{opt} and use this in (17) to design the $W_{RLDF}(\mathbf{x})$.

3. Project Description

The project folder contains a folder called “src”. This folder contains all the source files needed to work on the project or to be extended. The code has been written such that it could be easily adapted for parallel computing by MPI implementation, if one is interested to do so. However, for now you may skip anything related to MPI implementation. For this when you see anything called “MPI” in the program just ignore it. Inside “src” folder you will see a “Makefile”. In large projects, this file is created to make the compilation of project easier. Just google ”Makefile” and get more information on that.

In order to compile all the source files in your project, just go to src folder and using terminal just type “make” (on Linux base system including Mac OS) and hit enter. On windows if you have installed MingW and properly you have added the “bin” folder of MingW to the “PATH” environment variable, then when by terminal you switch to src folder, just type “migw32-make” and press enter. This should work as like “make” command on linux based systems. This is because when you install MingW, the migw32-make executable comes with it (in its bin folder). Nevertheless, the system doesn’t know where this migw32-make is so you have to add the path to the MingW bin folder to the PATH environment variable (if you have not done so yet on your windows machine). In any case, after entering “make” or “mingw32-make” commands, all source files in src folder will be compiled. Ignore any warning message. Once compiled, an executable file called “project” is created in “bin” folder. To run the project go to bin folder and type one of the commands that you need on terminal. Examples of commands and the options that you can pass to main file through command is written in “command.txt” file and “readme.txt” file that come with the project.

Folder “bin” contains four folders each of which containing a real dataset. if you go to each of these folders, there is a txt file. The first two lines in each txt file contains the total number of samples in the data and the total number of features. For example, on top of file “ChenLiver2004_10237x157.txt” we see a line containing 75 and 82 and the second line is 10237. This means that this file contains 157 sample points (75 from one class, 82 from another class) with 10,237 feature variables (here our features are genes). So in fact in this data we have had 157 individuals and for each of them we have found the amount of gene expression for each of 10,237 of their genes. This gives us a matrix of 157 columns and 10,237 rows. The content of this file (ChenLiver2004_10237x157.txt) is the values inside this matrix.

There are many details in this project package. Do NOT try to understand all the details. You need to understand the dependencies between files by looking at the contents and understand only what you need to implement/extend your project. Here is an example how you can get a general view of the project. For example, when you open main.cpp you will see that using parse_common_argument function (defined in parseArguments.cpp) we parse the command line. The effect of this command is to parse your command line. In fact when you go to bin folder and once you have the “project” executable in that folder, on terminal you can type a command like:

```
./project -i ChenLiver2004_10237x157/ChenLiver2004_10237x157.txt -o ChenLiver2004_10237x157/
Chen_output_60_10 -tr 60 -d 10 -mr 500
```

to run the project on “ChenLiver2004_10237x157.txt” and save the result in “ChenLiver2004_10237x157” folder. Now the way the program understands what each notation in your command line does, is through “parse_common_argument”. If you are interested in such applications try to understand this but for the sake of project you can simply ignore the details therein. Another example:

somewhere inside main.cpp you will see “calculateErrors(..)” function is called. So this means now the main function depends on calculateErrors function defined in calculateErrors.cpp. When you open calculateErrors.cpp you will see that this command/file depends on several other commands/files, e.g. dataGeneration.cpp. The effect of dataGeneration is to take some random subset of data of size “-tr” and save the data in (&*data_trn*) (in the command line above this data is taken from ChenLiver2004_10237x157.txt and has size 60) to build (train) a classifier and set aside the rest as sample points to test the accuracy of the classifier (&*data_tst*). You can check the program and see how “-tr” option in command line is passed to dataGeneration.cpp to generate a data of that size from ChenLiver2004_10237x157.txt.

Since the original number of features is very large (.e.g 10237 in ChenLiver2004_10237x157.txt), we need to work on a much smaller number of features (lets say 5 or 40 or ...). This is simply because the complexity of algorithms (SVM, LDA,...) increases as the number of features is increasing and, furthermore, many of these features are simply redundant. The feature selection step is achieved by "featureSelection" command in calculateErrors.cpp. You don't need to implement any feature selection method as this is being taken care of in the package. If you look at the programs you will see that eventually the "-d" option in the command line above is being passed to "featureSelection" command in calculateErrors.cpp to select a "d" number of features (in the example above 10).

If you continue investigating calculateErrors.cpp you will see that the classifiers (LDA, SVM with linear kernels (LSVM), SVM with radial kernels (RSVM)) are being built there on training data by "ldaTrn", "svmTrn". How are these classifiers built? These function are defined inside classifiers.cpp. Here again there are many details for example how svm classifiers are built. You are not supposed to understand them. For those of you who would like to "extend" the project (see Level 2, 3, and 4 extensions below), you may compare function ldaTrn with equation (1) above. In that case you will see that ldaTrn depends on "mean" and "cov" function, as formula (1) depends on them. Then check and see where/how these functions are defined in the program. Then you may also check ldaTst function in classifiers.cpp.

When a classifier is designed, it is important to know how well it performs in the future. By that we mean how well it can classify future sample points (that are not part of training set S^n) to the class that they are really coming from (class 0 or 1). When we build classifiers, we have to see how accurate they are. In calculateErrors.cpp, we estimate their accuracies in two ways. One is by using the test data (data_tst), which gives us a good estimate of what the "true" error of the classifier is on a set of independent data that was not used in training the classifier. In the project true errors for LDA, LSVM, and RSVM classifier are obtained by being saved in "(*all_errors).lda_true_error", "(*all_errors).lsvm_true_error", "(*all_errors).RSVM_true_error". See how they are obtained in calculateErrors.cpp (for example for LDA it is obtained from ldaTst function).

Another way, is to estimate the error of the classifier is by "error estimators" (0.632 bootstrap, cross-validation, resubstitution, bolstering, loo). These algorithms estimate the error by testing the accuracy on the same data that is used to train the classifier (training data). Since this project needs only the true errors for each classifier (not the error estimators), the first thing you need to do is to remove all redundant functions related to 0.632 bootstrap, cross-validation, resubstitution, bolstering, loo. For example, if you look at calculateErrors.cpp, we only need to keep "lda_true_error", "lsvm_true_error", "RSVM_true_error" for LDA, and SVMs (because they keep true errors) and remove anything related to "lda_resub_error", "lsvm_resub_error", lda_cvkfold_error and ..... .

Note that the -mr option in command line shows the number of simulations. By this we mean the process of randomly taking data from ChenLiver2004_10237x157.txt, constructing the classifier, and estimating its error whether by estimators or its "true" error is repeated "mr" number of times to give us a full view of the estimators and true error (gives us the distributional knowledge of how these are spread). Then we can find the expectation (average) performance of classifier by finding their average true error over mr randomly generated dataset.

3.1. Project: Which Classifier is Best in Small-Sample?

In this project we would like to check which classifier is most accurate in small-sample/large-dimension. We compare classifier in terms of average *true* error, not estimators. Again, as explained in the previous sections in this project we don't care about bootstrap or cross-validation, resubstitution, loo, bolserting estimators of error and you *must* remove the programs related to calculating these estimators from your project (*must* is explained later, points will be deducted if you don't do this). You need to run the program for various sample size n (-tr option in command line) and dimension p (-d option). While both types of SVM, DLDA, SEDC, RLDA, and QDA (the way we have defined it in (10)) can be built for situations that $n < p$ but LDA and G13 should be only built for cases that $n > p$. This is because the sample covariance matrix in (2) used to built LDA (and also G13) is singular and cannot be inversed if $n < p$. You need to choose some values of p and n, find the "true" error and properly compare classifiers together.

In the example in previous section, when I run the above command line, currently 18 output files will be created in ChenLiver2004_10237x157 folder (note that the package for now contain only LDA, LSVM, and RSVM). Just by looking at their names you will see that 6 files belong to LDA, 5 of which are the results of 5 estimators (0.632 bootstrap, resubstituion, 5 fold cross-validation, leave-one-out, and bolster estimators) and 1 file is the result of true error of LDA. You *must* modify the program in such a way that it does NOT calculate the estimators and does NOT produce these redundant files: "Chen_output_60_10_lda_bolster_error_round=1", "Chen_output_60_10_lda_boot632_error_round=1",

"Chen_output_60_10_lda_cvkfold_error_round=1", "Chen_output_60_10_lda_loo_error_round=1",

and "Chen_output_60_10_lda_resub_error_round=1", or removing similar estimators files for SVMs. The only files you need to work with are files of true error, e.g., "Chen_output_60_10_lda_true_error_round=1", or "Chen_output_60_10_lsvm_true_error_round=1", or "Chen_output_60_10_RSVM_true_error_round=1" because these files keep the result of true error of classifiers and we can use them to compare classifiers together.

Now if you open each of these files, there are 500 numbers (-mr option in command line). One way you can compare the classifiers is to take the average of this 500 numbers in each "true" error file. The lower the average (Expectation) of the true error is, the better the performance of that classifier for that dimension and sample size (in the example above for p=10, and n=60).

Now to compare the classifier we have to compare them over a range of sample size and dimensions. To do so, we can run the project for various sample size and dimension and plot the curve of expectation of true error versus sample size for each dimension. You need to run the project for sample size $n = 40, 60, 80, 100, 120$ and for each sample size consider dimensions $p = 3, 5, 10, 40, 80$, save the results, and then for each dimension plot a curve of Expectation of true error versus sample size. Remember that as we said before the results of LDA and G13 should not be compared for cases where sample size (-tr option) is smaller than dimension (-d option). The students need to properly manage the output files of the project whether manually or automatically for various dimensions and sample sizes, write a report, and submit the report. The conclusion of which classifier is better should be properly supported by their graphs and/or tables. The plots can be produced by any software including R, MATLAB, ... Figure 1 shows an example of a plot only for $p = 5, 20$. But there are ways you can even improve this plot but it is up to you anyways.

For example in this plot (Figure 1) we compare several classifiers together. The plots show the expectation of true error of classifiers versus the sample size n. Although I have explained the label of x axis (sample size) or y axis (Expectation of true error) in the figure caption, but we could have explicitly add them for each figure to the plot. If these were the real plots we were getting, we could judge that the SEDC classifier is almost the best classifier here. This is because the average true error of this classifier was less than others for various dimensions.

The report should have an abstract, introduction, a section called "Systems and Methods" in which the students need to describe the classifiers that they use (you can use the content of this file but you need to cite proper references (journal articles/books describing the methods, do not "cite" this file)), and a section called "Results and Discussion" in which the students describe their results, put plots, and draw a conclusion. Note that all the projects/results should be run on all four datasets in the "bin" folder. When you make a conclusion, you need to look at your results on all four datasets and make a conclusion which is more or less consistent on all datasets.

Below I describe various projects that the students may choose to work on. Students are supposed to work in teams of 5 and by Friday October 28th, students need to finalise their team members and add the name to google doc that I will share. After this date, the students may not change their teams. Each team may choose to work on one of the following groups. The level of difficulty of the project increases from level 1 to 4. While the students don't need to report in advance which group they have chosen to work on, but in their final report they need to clearly state to which level their project belongs. I will check this claim by your report and your codes that you submit. In level 2, 3, and 4 below you need to extend the existing codes of the project. If you claim that your project belongs to one level but I realize that this is a false claim (through your codes/reports), you will receive negative points (-40 out of 100). The grading style described below belong to 25% project report/code. The 10% project presentation is totally separate from this. You

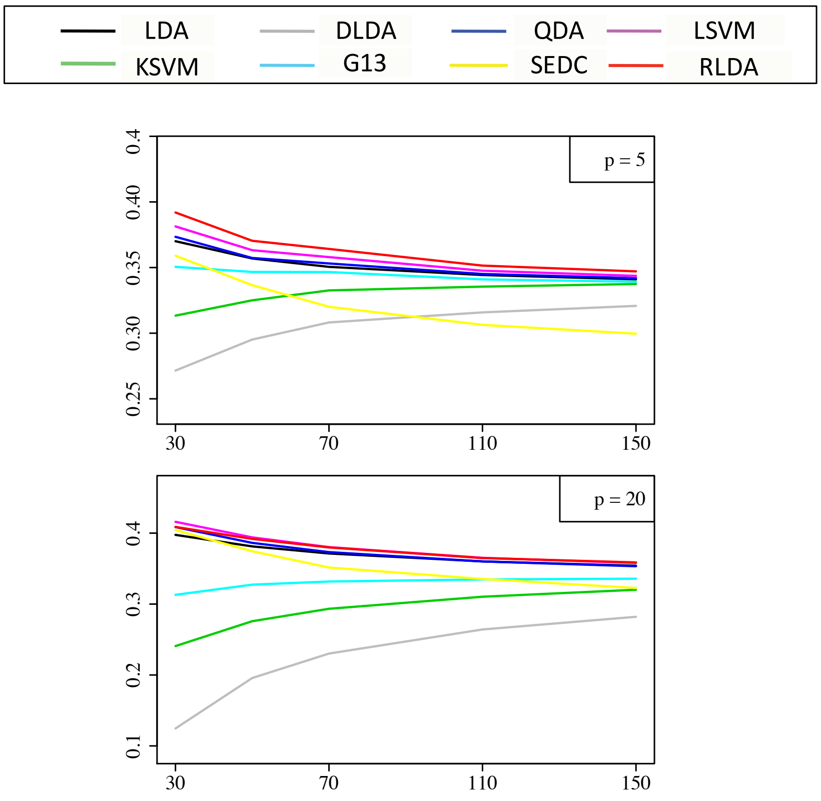

Figure 1: Performance of classifiers versus sample size (n) for different dimensions (p). The y-axis in each plot presents the expectation of true error. The x-axis presents the total number of samples in both classes. This figure is only for illustration purposes and don't draw any conclusion based on this figure.

may receive a bad score on your report for the project but a perfect score on presentation or vice-versa. The instructions for presentations will be announced later.

NOTE: In all simulations use and compare classifiers for these sample sizes and dimensions:

$n = 40, 60, 80, 100, 120$

$p = 3, 5, 10, 40, 80$

Level 1: No Extension to the project: In this module, the students run the project by properly managing the output files of the project whether manually or automatically for various dimensions and sample sizes, write a report, and submit the report. Normally, in this module the only attempt is to remove all redundant functions in the source codes in the project package. The students can use the package as it is (remove redundancies) and run the results. Note that since LDA is not applicable to situations where $p > n$ then you should not have its result for those situations. In your output, you need to plot your results for $n = 40, 60, 80, 100, 120$ and $p = 3, 5, 10, 40, 80$. Then for example your plot of expectation of true error for $p = 10$ should have results of LSVM, RSVM, and LDA in situations for the whole range of n $p = 3, 5, 10, 40, 80$ because in any case $n > p = 10$. However, your plot for p=80, should contain both type of SVMs for n=40 to 80 (because $n <= p$) but LDA and both SVMs results for n=100 to 120 (because in this range $n > p$).

Level 1 Grading: The "best" project in this module receives 70 points out of 100 total grade for the project. The best project is a project in which the conclusion is best supported by the results and plots and redundant functions related to error estimators are removed (not related to true error). Having plot in your project helps understating your project better. Note that having a nice report itself is also an important factor in your final project grade.

Level 2: First Round Extension: In this module, the students first add the Diagonal Linear Discriminant Classifier and/or G13 and/or QDA (section 2.2, 2.3, and 2.5 above) to the project package. Then the students run the project by properly managing the output files of the project whether manually or automatically for various dimensions and sample sizes, write a report, and submit the report. Note that unlike LDA and G13, the results of DLDA and QDA are valid not only for cases where $p < n$, but also for situations where $p > n$. So now your results for situations where $p < n$ should have linear kernel SVM, radial kernel SVM, LDA, DLDA, G13, and QDA and as soon as $p > n$ they should include only SVMs, QDA, and DLDA.

Level 2 Grading: The "best" project in this module receives 95 points out of 100 total grade for the project. The best project is a project in which the conclusion is best supported by plots. Having plot in your project helps understating your results (and having a better grade). By properly writing/integrating DLDA to the project you get 6 additional point w.r.t Group 1. By properly writing/adding G13 you will also receive 6 additional points. By properly writing/adding QDA you will receive 13 additional points. This means that if you report is comparable to best report from Level 1, then by "properly" adding these three classifiers to the project and your report you will receive 95 out of 100. Nevertheless, this doesn't guarantee that a project in this group will receive a better grade than any project in Level 1 group. A weak project report in this group may receive a worse grade than a strong report from Level 1 group. Note that in Level 1 you had to also remove redundant functions related to error estimators such as bootstrap, cross-validation, loo, resubstituiton. This should be also done in this level because for going to any higher level extension, you should completely perform the previous level tasks. Otherwise, it will not be accepted.

Level 3: Second Round Extension: In this module, the students first add the Diagonal Linear Discriminant Classifier AND G13 AND QDA (Expected tasks in Level 2) to the project package. However, in addition to what Level 2 groups, the students will add the results of SEDC in their project for $*k = 1*$

(see section 2.6 for definition of k for SEDC). Note that if you compare your results for any other k rather than 1 you will not get any point even if you have coded everything properly. Then the students run the project by properly managing the output files of the project whether manually or automatically for various dimensions and sample sizes, write a report, and submit the report. Note that the results of SEDC will be valid regardless of $p > n$ or $n > p$.

Level 3 Grading: The "best" project in this module receives a 105 points out of 100 total grade for the project. This means that 10 additional points w.r.t. to Level 2 will be considered for the best project in this group. However, for going to this level, students are expected to completely implement Level 2. In addition, going to this level again doesn't mean the project will receive a higher grade than a project in Level 2. A weak project report in this group may receive a worse grade than a strong report from Level 1 and/or Level 2 groups.

Level 4: Third Round Extension: In this module, in addition to anything that was added by level 1, level 2, and level 3, the students will add RLDA classifier (see section 2.7) and add its results to all plots. Note that the results of RLDA are valid for both $n > p$ or $n < p$ because we do not have singularity problem anymore as in LDA or G13 that use simply sample covariance matrix.

Level 4 Grading: The "best" project in this module receives a 115 points out of 100 total grade for the project. Adding RLDA must be in addition to what students in level 3 do to be eligible for receiving 115. Adding RLDA without considering what was described in level 2 and 3 will not be accepted at all.

[1] T. W. Anderson, *Introduction to Multivariate Statistical Analysis.* New York: Wiley, 1958.

[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification.* New York: Wiley, 2000.

[3] V. I. Serdobolskii, "Discriminant analysis of high-dimensional data over limited samples," *Dokaldy Mathematics*, vol. 81, pp. 75–77, 2010.