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 2015 (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, Error Estimation, 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)

well-known 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

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

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

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 LDF *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

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 (5) 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. Euclidean Distance Classifier

Euclidean distance classifier (EDC) is also closely related to LDA with the difference that there is no sample covariance matrix involved in the classifier. EDC is defined by the euclidean distance function which is defined by

$$W_{EDC}(\mathbf{x}) = \left(\mathbf{x} - \frac{\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1}{2}\right)^T (\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}_1)\,. \tag{7}$$

Similarly, the classifier is then

$$\psi_{EDC}(\mathbf{x}) = \begin{cases} 1\,, & \text{if } W_{EDF}(\mathbf{x}) \leq \mathrm{c} \\ 0\,, & \text{if } W_{EDF}(\mathbf{x}) > \mathrm{c} \end{cases}, \tag{8}$$

2.4. 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

$$\begin{aligned} &\min_{\mathbf{v},c} ||\mathbf{v}|| \\ &\text{s.t.} \\ &y_i(\mathbf{v}^T\mathbf{x_i} - c) \geq 1 \end{aligned} \tag{9}$$

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].

2.5. k-NN

The concept behind this classifier is simple. A sample point $\mathbf{x}$ will be classified to either class 0 or 1 if the majority of the k nearest neighbors of $\mathbf{x}$ among training sample points (i.e. among S^n) belong to that class. Formally we can define this classifier as follows:

$$\psi^{kNN}(\mathbf{x}) = \begin{cases} 1\,, & \text{if } \sum_{i=1}^{n} w_{ni} I_{Y_i=1} > \sum_{i=1}^{n} w_{ni} I_{Y_i=0} \\ 0\,, & \text{otherwise} \end{cases}, \tag{10}$$

where $w_{ni} = 1/k$ if $\mathbf{x}_i$ is among the k nearest neighbors of $\mathbf{x}$, and $w_{ni} = 0$ elsewhere, and I_A is the indicator function; that is when event A happens, $I_A = 1$, and otherwise, $I_A = 0$. $\mathbf{x}_i$ is said to be the k-th nearest neighbor of $\mathbf{x}$ if the Euclidian distance $||\mathbf{x} - \mathbf{x}_i||$ is the kth smallest among $||\mathbf{x} - \mathbf{x}_1||$, ..., $||\mathbf{x} - \mathbf{x}_n||$. In case of tie, candidate with the smaller index is said to be closer to $\mathbf{x}$.

3. Error Estimators

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). For this purpose we have to estimate the error of a classifier. Some popular error estimators are being described here.

3.1. Apparent Error or Resubstitution

The fastest and simplest way for estimating the error of a classifier (when no test data is available) is to see how correctly the classifier can classify the training sample points to either class 0 or 1 [3]. Mathematically, for a classifier $\psi(\mathbf{x}_i)$ (this could be any classifier including $\psi_{LDA}(\mathbf{x})$, $\psi_{DLDA}(\mathbf{x})$, $\psi_{EDC}(\mathbf{x})$, $\psi^{SVM}(\mathbf{x})$, or $\psi^{kNN}(\mathbf{x})$) this can be represented as follows

$$\hat{\epsilon}_{resub} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \psi(\mathbf{x}_i)|. \tag{11}$$

However, this estimator is generally optimistic, meaning that what it estimate as the error of the classifier in in fact less than the actual error of the classifier.

3.2. Cross-validation

In k-fold cross-validation, the training sample S^n is randomly partitioned to k folds $F_{(l)}$, for $l = 1, 2, ..., k$ (for simplicity assume k divides n). Each fold is left out one at a time, a classifier is constructed using the remaining training samples, the accuracy of this classifier is tested by using the fold that is left out (just testing how many errors the classifier committees on the held-oh fold), and at the end the overall proportion of errors committed on all folds will be the estimated error. This can be mathematically characterized as follows:

$$\hat{\epsilon}_{cvk} = \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{n/k} |y_j^{(i)} - \psi^{S^n \setminus F_{(i)}}(\mathbf{x}_j^{(i)})|. \tag{12}$$

where $(\mathbf{x}_j^{(i)}, y_j^{(i)})$ is a sample in the ith fold, and the super index in $\psi^{S^n \setminus F_{(i)}}$ shows that the classifier is designed using samples in S^n that are not part of $F_{(i)}$, i.e. $S^n \setminus F_{(i)}$. This process may be repeated several times. Each time the data is portioned differently, the cross-validation estimate is calculated, and the results will be averaged. A special case of k fold cross-validation is the case where $k = n$. In this case a single observation is left out each time, the classifier is constructed using the remaining sample points, and will be used to classify the sample that is left out. Mathematically this represented as

$$\hat{\epsilon}_{loo} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \psi^{(i)}(\mathbf{x}_i)|. \tag{13}$$

3.3. 0.632 Bootstrap

If we draw with replacement n samples from S^n, we obtain the so-called "bootstrap-sample", which we denote by $S^{n,*}$. Therefore, some of the samples in S^n will appear multiple times in $S^{n,*}$, whereas others will not appear at all. In bootstrap estimation we repeat the process of drawing n samples with replacement B times, obtaining B set of bootstrap samples denoted by $S_1^{n,*}$, $S_2^{n,*}$, ..., $S_B^{n,*}$. Then in order to obtain 0.632 estimator we first need to compute $\hat{\epsilon}_0$ estimator. $\hat{\epsilon}_0$ counts the number of those training samples misclassified which didn't appear in the bootstrap sample $S_1^{n,*}$, $S_2^{n,*}$, ..., $S_B^{n,*}$. As such $\hat{\epsilon}_0$ is obtained by summing the misclassified samples over all B bootstrap samples and dividing the sum by the total number of training samples not appearing in the B bootstrap sample. Let $A_b = \{i | \mathbf{x} \notin S_b^{n,*}\}$. In other words, A_b is the set of indecis of samples that don't appear in $S_b^{n,*}$. Mathematically, $\hat{\epsilon}_0$ is obtained by

$$\hat{\epsilon}_0 = \frac{\sum_{b=1}^{B}\sum_{\mathbf{x}_i \in A_b} |y_i - \psi^{S_i^{n,*}}(\mathbf{x}_i)|}{\sum_{b=1}^{B} |A_b|} \tag{14}$$

Having $\hat{\epsilon}_0$ then the 0.632 bootstrap estimator will be

$$\hat{\epsilon}_{b632} = (1 - 0.632)\hat{\epsilon}_{resub} + 0.632\hat{\epsilon}_0, \tag{15}$$

where $\hat{\epsilon}_{resub}$ is defined in (11).

3.4. 0.632+ Bootstrap

This estimator is proposed in [4] and is closely related to 0.632 bootstrap. Let $\hat{p}_1$ be the observed proportion of samples that are coming from class 1 (i.e. $\frac{n_1}{n}$), and $\hat{q}_1$ be the observed proportion of samples that are classified as if they belong to class 1. Then define $\hat{gamma}$ such that

$$\hat{\gamma} = \hat{p}_1(1 - \hat{q}_1) + (1 - \hat{p}_1)\hat{q}_1. \tag{16}$$

Then let $\hat{\epsilon}'_0$ be defined as

$$\hat{\epsilon}'_0 = \min(\hat{\gamma}, \hat{\epsilon}_0) \tag{17}$$

where $\hat{\epsilon}_0$ is defined in (14). In addition, define $\hat{R}'$ such that

$$\hat{R}' = \begin{cases} R\,, & \text{if } R \in [0,1] \\ 0\,, & \text{otherwise,} \end{cases} \tag{18}$$

where $R = \frac{\hat{\epsilon}_0 - \hat{\epsilon}_{resub}}{\hat{\gamma} - \hat{\epsilon}_{resub}}$. Then

$$\hat{\epsilon}_{b632+} = \hat{\epsilon}_{b632} + (\hat{\epsilon}'_0 - \hat{\epsilon}_{resub}) \frac{0.368 \times 0.632 \hat{R}'}{1 - 0.368 \hat{R}'}. \tag{19}$$

4. 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 10 or 30 or ...). This is simply because the complexity go 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, SVM with radial kernels) 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 (5) above. In that case you will see that ldaTrn depends on "mean" and "cov" function, as formula (5) depend on them. Then check and see where/how these functions are defined in the programm. Then you may also check ldaTst function in classifiers.cpp.

As we said before, 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. Another way, is to estimate the estimators that we explained in previous section (0.632 bootstrap, cross-validation, resubstitution,...) to see what these estimators suggest by testing the accuracy on training data. By comparing what these estimators suggest with the "true" error, we can see which estimator is closer to true error meaning that which estimator performs better in practice. This is important to investigate because generally in practice (small-sample situations) we use all the available data for building the classifier so there is nothing left to find the "true" error and the only option we have is to use estimators such as bootstrap, cross-validation, ... to estimate the accuracy of classifier. Now we have to see which estimator is better in practice. 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 the estimators relative to true error.

4.1. Project: Bootstrap versus Cross-validation in Small-Sample

Here we would like to compare the performance of bootstrap estimator with cross-validation estimator (5-fold cross-validation and leave-one-out). If you look at k-fold cross-validation formula in (13) and compare them with the project files that we have (with calculateErrors.cpp) you will see that the default value of "k" parameter in CV is 5 in our program (int $K = 5$ in calculateErrors.cpp). Similarly, you will see that the value of "B" parameter in 0.632 bootstrap in formula (15) is 100 in our programs. Note that a "complete" project in each of group that we describe later, should have at least the results of 0.632 bootstrap, 5-fold cross-validation, and leave-one-out (loo). This is because loo is a special type of cross-validation.

In this project we would like to check which estimator is more accurate in small-sample/large-dimension. Bootstrap or cross-validation (5-fold and leave-one-out)? The package that you have also includes some other estimators such as resubstituion and bolstering but we don't care about them. 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 can be built for situations that $n < p$ but LDA should be only built for cases that $p > n$. You need to choose some values of p and n, run the estimators and at the same time find the "true" error and properly compare them together. In the example in previous section, when I run the above command line, 18 output files will be created in ChenLiver2004_10237x157 folder. 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 another estimator known as bolster estimator) and 1 file is the result of true error of LDA. You simply ignore result of resubstituiton and bolserting. Now if you open each file, there are 500 numbers (-mr option in command line). One way you can compare the estimators is to take the average of this 500 numbers in each "estimator" file and compare with the average of the corresponding numbers in "true_error" file. The closer the average (Expectation) of the estimator is to the true error, the better the performance of that estimator. However, average (mean) is only one metric of performance. Another metric is the Root Mean Square Error (RMS), which is defined by $\mathrm{RMS}(\hat{\varepsilon}) = \sqrt{\mathrm{E}[(\hat{\varepsilon} - \varepsilon)^2]}$, where we denote the true error and the estimator by ε and $\hat{\varepsilon}$, respectively. The RMS is decomposed into the bias, $\mathrm{Bias}(\hat{\varepsilon}) = \mathrm{E}[\hat{\varepsilon} - \varepsilon]$, of the error estimator relative to the true error, and the deviation variance, $\mathrm{Var}_{\mathrm{dev}}(\hat{\varepsilon}) = \mathrm{Var}(\hat{\varepsilon} - \varepsilon)$, by

$$\mathrm{RMS}(\hat{\varepsilon}) = \sqrt{\mathrm{Var}_{\mathrm{dev}}(\hat{\varepsilon}) + \mathrm{Bias}(\hat{\varepsilon})^2}. \tag{20}$$

Therefore, to compute the RMS you need to compute the above metric by taking the values of the estimators and true error from the output files. For example to compute the "Bias" term, you can look at the output for estimators and true error for the same run, find their differences (the difference of estimator from true error for the same run out of 500 runs), and find the average of differences across the 1000 runs. To compute the variance of deviation, you can do the same but finding the variance of differences rather than taking the mean (average) of differences. Then you can add them up (Bias and variance of deviation) and take the square root to find the RMS.

Now to compare the bootstrap and cross-validation (5-fold cross-validation and leave-one-out), we can run the project for various sample size and dimension and plot the curve of Expectation and/or RMS versus sample size for each dimension. For example, one may run the project for sample size n=30, 50, 70, 90, 100 and for each sample size for dimension p=10, 20, 50, 100 or 150, save the results, and then for each dimension plot a curve of Expectation and/or RMS versus sample size. Nevertheless, these are just examples and you may choose to change/add various sample size and/or dimension depending on the run time and/or your interests. But remember that as we said before the results of LDA should not be run for case where sample size (-tr option) is larger 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. Whether bootstrap is better than cross-validation or cross-validation is better than bootstrap 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. 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 estimators together. The plots on the left columns show the expectation of the estimators and the plots on the right column show the RMS. Note that the RMS plots do not have the black curve (the true error) because RMS is an objective and relative performance metric to see how close the estimator is to true error. In other words, the information of true error is embedded inside the RMS metric. However, the expectations on the left column is a subjective measure in the sense that we have to look at the curve of Expectation for each estimator and compare it with the curve of true error and see how close they are. Although we have put the label of x (sample size) or y (Expectation or RMS) axes in the figure caption, but we could have explicitly add them for each figure to the plot.

The report should have an abstract, introduction, a section called "Systems and Methods" in which the students need to describe the classifiers/estimators 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)),

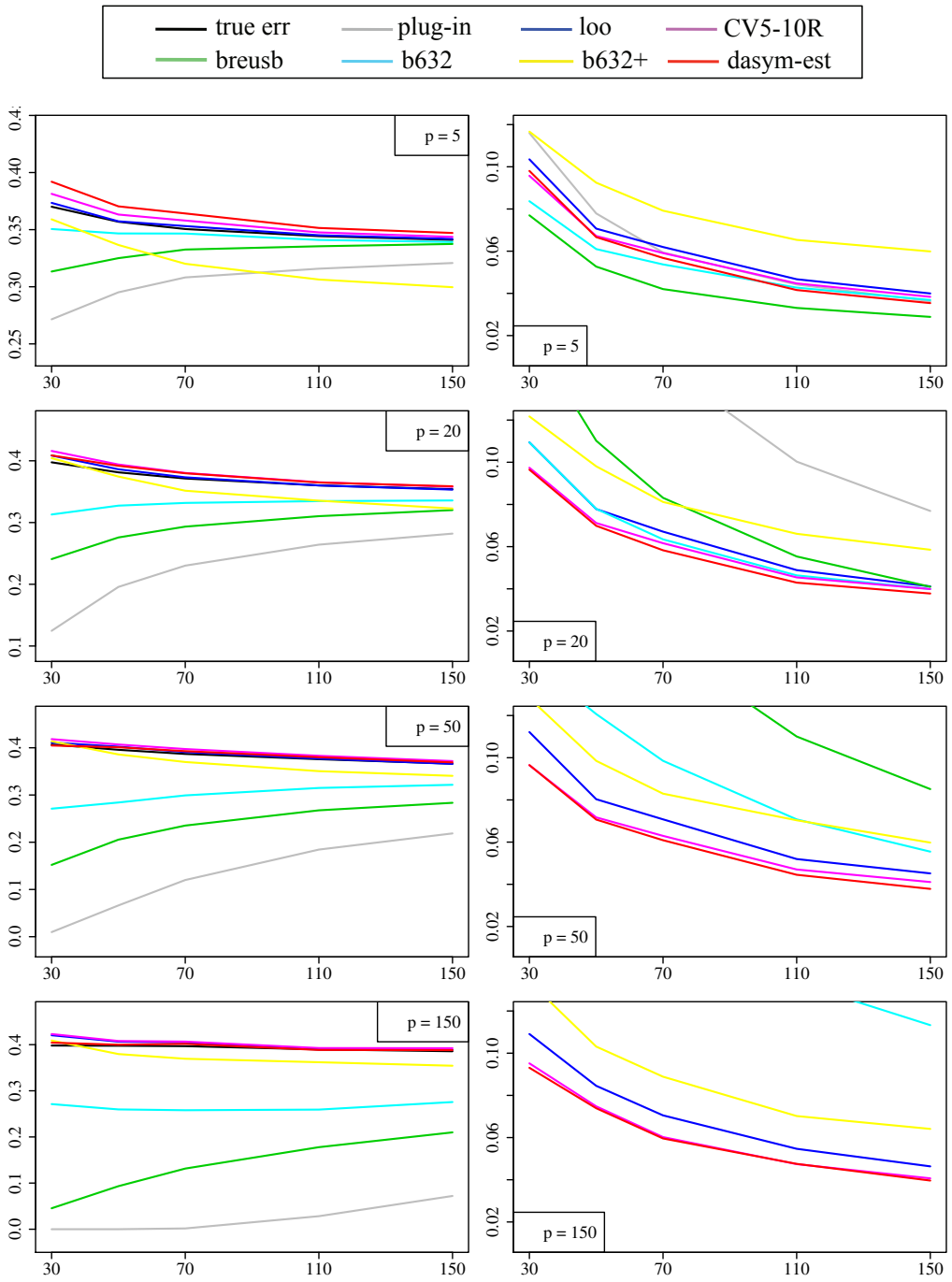

Figure 1: Performance (Expectation and RMS) of estimators of true error of RLDA classifier. The y-axis in the left column of each plot presents the expectations and in the right column presents RMS. The x-axis presents the total number of samples in both classes. Due to large RMS of some of the estimators, their RMS curves are partially or entirely removed from the plots. p denotes dimension.

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 4 or 5 and by Monday November 2nd, students need to finalize their team members and add the name to moodle. 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 group 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 which group their project belongs to. I will check this claim by your report and your codes that you submit. In group 2, 3, and 4 below you need to extend the existing codes of the project. If you claim that your project belongs to group 3 but I realize that this is a false claim (through your codes/reports), you will receive negative points (-20 out of 100). The grading style described below belong to 25% project report/code. The 10% project presentation is totally separate from this. You may receive a bad score on your report for the project but a perfect score on presentation or vice-versa. For the presentations I will ask questions from each member of the group and I will grade them individually and based on presentation. The schedule for presentations will be announced later.

Group 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 no attempt is made to change the content of source files in the project package. The students can use the package as it is 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. Lets say you choose n=30, 50, 70, 90, 100 and p=10, 20, 50, 100 or 150. Then for example your plot of expectation for $p = 10$ should have bootstrap and cross-validation (5-fold and loo) for both type of SVM and LDA in situations for the whole range of n (n=30, 50, 70, 90, 100) because in any case n¿p=10. However, your plot for p=50, should contain both type of SVMs for bootstrap and cross-validation for n=30 to 50 (because $n <= p$) but LDA and all SVM results for n=70 to 100 (because in this range $n > p$).

Group 1 Grading: The "best" project in this module receives 80 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. Having plot in your project helps understating your project better. In addition, in order to receive a grade of 80, you need to have both metrics of comparisons, (Expectation and RMS) results/plots.

Group 2: Extension of the project–level1 Extension: In this module, the students first add the Diagonal Linear Discriminant Classifier and/or Euclidean Distance Classifier (section 2.2 and 2.3 above) to the project package. They properly add them and integrate them with cross-validation, 0.632 bootstrap, leave-one-out (loo). 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, the results of Diagonal Linear Discriminant Classifier and/or Euclidean Distance Classifier are valid not only for cases where $p < n$, but also for situations where $p > n$. So one goal of performing level 1 extension, is to compare bootstrap and cross-validation for more classifiers for situations in which $p > n$ (because otherwise, we only compare them using linear kernel SVM and radial kernel SVM). Nevertheless, the result of LDA (and all other classifiers) should be present for situations where $p < n$.

Group 2 Grading: The "best" project in this module receives 100 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 Diagonal Linear Discriminant Classifier to the project you get 10 additional point w.r.t Group 1. By properly writing/adding Euclidean Distance Classifier you will also receive 10 additional points. This means that if you report is comparable to best report from Group 1, then by "properly" adding these two classifiers to the project and your report you will receive 100 out of 100. Nevertheless, this doesn't guarantee that a project in this group will receive a better grade than any project in group 1. A weak project report in this group

may receive a worse grade than a strong report from group 1.

Group 3: Extension of the project–level2 Extension: In this module, the students first add the Diagonal Linear Discriminant Classifier and/or Euclidean Distance Classifier (similar to Group 2) to the project package. They properly add them and integrate them with cross-validation, 0.632 bootstrap, and leave-one-out (loo). However, in addition to what group 2 does, the students will add the results of 0.632 bootstrap plus (section 3.4) to all classifiers that they have in their project. 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. In this way, they can compare the two forms of bootstrap (0.632 or 0.632+) with the two forms of cross-validation (5-fold and loo) and using several classifiers (both SVMs, Diagonal LDA, Euclidean Distance classifier (regardless of $p > n$ or $n > p$) and both SVMs, Diagonal LDA, Euclidean Distance classifier and LDA for cases where $p < n$.

Group 3 Grading: Extension of the project–level3 Extension: The "best" project in this module receives a 115 points out of 100 total grade for the project. This means that 15 bonus points will be considered for the best project in this group. By properly writing/integrating Diagonal Linear Discriminant Classifier to the project you get 10 additional point w.r.t Group 1. By properly writing/adding Euclidean Distance Classifier you will also receive 10 additional points. This means that if you report is comparable to best report from Group 1, then by "properly" adding these two classifiers to the project and the results of your report you will receive 100 out of 100. However, by adding bootstrap 0.632+ to "all" classifiers in the project you will receive 15 additional points. This means that if your project is comparable to the best project in group 2, you will receive 115 points by properly adding the 0.632+ to all classifiers. Nevertheless, this doesn't guarantee that a project in this group will receive a better grade than any project in group 1 or group 2. A weak project report in this group may receive a worse grade than a strong report from group 1 and/or group 2.

Note: One way decide to add only bootstrap 0.632+ without adding Diagonal Linear Discriminant Classifier and/or Euclidean Distance Classifier . In that case, this is considered as a Level 2 Extension and adding 0.632+ to the existing classifiers in the project will have 10 additional points. Therefore, adding 0.632+ to the current classifiers of the project you may receive at most 90 points out of 100. Adding 0.632+ and one of the classifiers Diagonal Linear Discriminant Classifier or Euclidean Distance Classifier (not both) will have at most 100 out of 100.

Group 4: Extension of the project–level3 Extension: In this module, in addition to anything that was added by group 1, group2, and group 3, the students will add k-NN classifier (see section 2.5) and integrate it with all estimators including 0.632+ Bootstrap.

Group 4 Grading: Extension of the project–level3 Extension: The "best" project in this module receives a 125 points out of 100 total grade for the project. Adding k-NN must be in addition to what students in group 2 and 3 do to be eligible for receiving 125. Adding k-NN without considering what was described in level 2 and 3 will be considered a level1 extension and has 10 additional point w.r.t. to level1 extension.

References

[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] S. Raudys, *Statistical and Neural Classifiers, An Integrated Approach to Design.* London: Springer-Verlag, 2001.

[4] B. Efron, "Improvements on cross-validation: The .632+ bootstrap method," *J. Am. stat. Assoc.*, vol. 92, p. 548560, 1997.