Classifiers

Group 25
“Level 4 Has been Completely Implemented”

Abyzbay Diyar 100 “Second Student in the Presentation”
Bizhigit Talgat 100 “Fifth Student in the Presentation”
Bolat Khaknazar 100 “First Student in the Presentation”
Dairabay Zhumazhenis 100 “Fourth Student in the Presentation”
Tokmyrza Batyrkhan 100 “Third Student in the Presentation”

Programming for Engineers BENG 146
Instructor: Amin Zollanvari
Nazarbayev University
Fall 2016

PROJECT (LEVEL 4, full completed)

1. ABSTRACT:

This project aimed to analyze various types classifiers, see which classifier is best for different situations, and find the classifier that would produce most accurate results. LDA, LSVM, KSVM, G13, QDA, DLDA, SEDC and RLDA were explored in this project. Each classifier was described and each of their outputs was illustrated by means of graphs and tables. The graphs showed the dependence of average true error on data of various sizes and dimensions. The classifiers were compared and the most efficient one was found by analyzing those graphs.

2. INTRODUCTION:

A lot of methods for extracting information from various types of data were developed by many researchers who specialized in signal processing and statistics during the past fifty years. Most of the widely known methods in this area were made for instances where a dataset is of a large sample size for fewer variables is available. However, nowadays we have an entirely different situation than what those methods were developed for; many practical situations involve large number of variables and datasets with small number of samples. That is why in general the performance of those well-known methods is required to be re-evaluated and changed to suit our needs, especially pattern recognition. The purpose of pattern recognition is to automatize the process of finding specific patterns in a dataset by developing the set of mathematical, statistical and computational methods. Lots of mathematical models like classifiers, error estimators, feature selections were developed during the past few decades. Some widely known mathematical models in this field are described below.

3. SYSTEM AND METHODS:

3.1. Discriminant analysis:

Discriminant analysis predicts membership in a class based on a several features that describe the object. Specifically, discriminant analysis predicts that, what class(in our case, 0 or 1) a classification X vector with the given(reduced, not all) number of features belongs to. As discriminant analysis can be achieved using some(not all) features, it is very useful, and it has high performance, when the data consists of thousands of features. The essential thing in discriminant analysis is to use a training data best by applying the best set of features according to sample size and feature size that is needed. A training data is built with the help objects with known class and features. Also, the best set of features, that is going to best identify class membership, will be chosen from whole features according to the training data (Teknomo 2015). Discriminant analysis is widely used in bankruptcy prediction, face recognition, marketing, biomedical studies, earth science. During the project, how the set of best features were chosen will not be covered, but its code was provided already.

Simple explanation of discriminant analysis: Suppose that, there are 2 types(A, B) of apples in the box, and there are 20 apples(A type), 20apples(B type). Type is usually class. And we have a database of apples that consists their type and 50 features about them: color, size, taste, … … … Then we pick any apple from the box, and we want to identify the type of the picked apple. But, we want to identify it fast, without considering all features of apple. To do so, we build the training data with the help of another 22(sample size for training data, n) apples with known type. Suppose that, set of best features consists of color, size, taste. And then it predicts the type of the apple according to color, size, taste. (Table for illustration purposes).

	Type A				Type B			
Name: Features:	Apple1	Apple2	…	Apple20	Apple21	Apple22	…	Apple40
Color:	C1	C2	…	C20	C21	C22	…	C40
Size:	S1	S2	…	S20	S21	S22	…	S40
Taste:	T1	T2	…	T20	T21	T22	…	T40
Other1:	#1	#2	…	#20	#21	#22	…	#40
Other2:	#1	#2	…	#20	#21	#22	…	#40
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
Other47:	#1	#2	…	#20	#21	#22	…	#40
C, S, T are numbers, which represent color, size, taste values respectively								

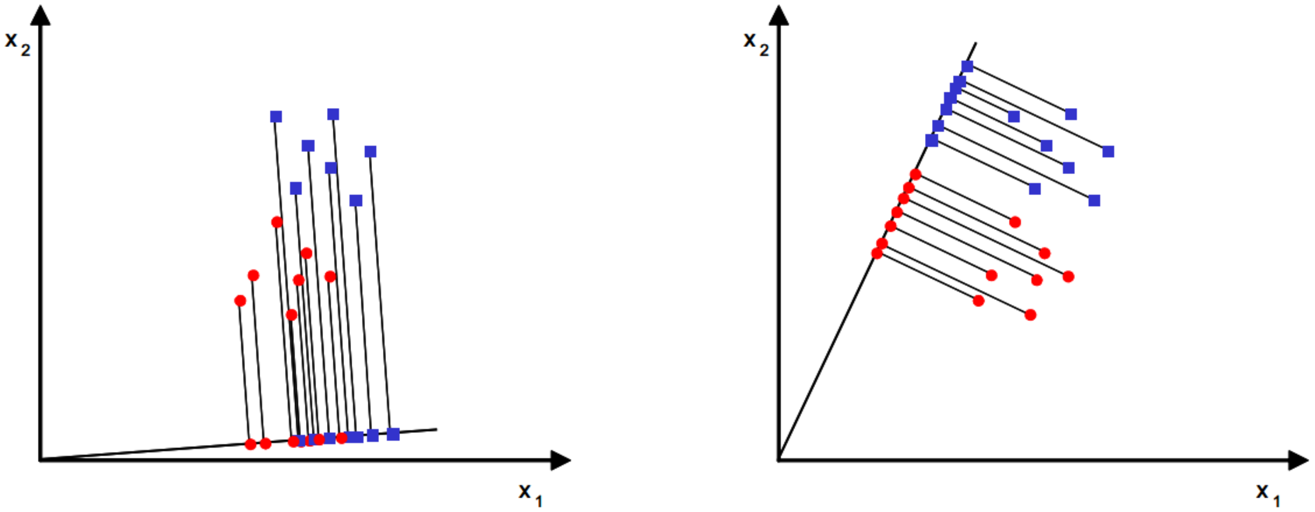
3.2. Classifiers:

6 Discriminant Analysis and 2 SVM classifiers were used in our project:

1. LDA - Linear Discriminant Analysis
2. QDA - Quadratic Discriminant Analysis
3. DLDA - Diagonal Discriminant Analysis
4. G13
5. RLDA - Regularized Linear Discriminant Analysis
6. SEDC by Serdobolskii
7. SVM: LSVM, KSVM.

3.2.1. Linear Discriminant Analysis

The purpose of LDA is to reduce dimensionality at the same time preserving as much of the class discriminatory information as possible. It tries to determine directions that best separates different class objects (Gong, Webb and Duda, n.d.). LDA originally was developed in 1936 by R.A. Fisher for two class problem.



Graph 1: Separation the scalars by a projection (Ricardo, Guitierrrez-Osuna n.d.).

In the first graph in Graph 1 objects are not separated well on the directed line, while in the second graph objects are best separated using LDA, considering two features X_1 and X_2 of the object.

If three features are considered, then objects are separated by the plane. If more than three features are considered, then objects are separated by hyper-plane (Teknomo 2015).

Prediction of the class is achieved considering the features of the object (object is x vector in formula) according to the formula below (Zollanvari 2010):

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

where $\bar{x}_0 = \frac{1}{n_0}\sum_{x_l \in S_0^n} x_l$ and $\bar{x}_1 = \frac{1}{n_1}\sum_{x_l \in S_1^n} x_l$ are sample means of 0 and 1 classes. C is pooled sample covariance matrix

$$C = \frac{(n_0 - 1)C_0 + (n_1 - 1)C_1}{n_0 + n_1 - 2}$$

where C_i is covariance matrix of each class where $i = \{0,1\}$

$$C_i = \frac{1}{n_i - 1}\sum_{x_l \in S_i^n} (x_l - \bar{x}_i)\,(x_l - \bar{x}_i)^T$$

The LDA classifier is

$$\psi_{LDA}(x) = \begin{cases} 1, & if\ W_{LDF}(x) \le c \\ 0, & if\ W_{LDF}(x) > c \end{cases}$$

where $c = \log\frac{N_1}{N_0}$.

LDA produces at most N − 1 feature projections, because calculating the inverse of covariance matrix is needed, but if d≥N covariance matrix is not invertible (Ricardo, Guitierrrez n.d.).

3.2.2. Quadratic Discriminant Analysis

In LDA covariance matrices of both population are pooled. But, in QDA covariance matrices is not pooled, and QDA is implemented using the following formula:

$$W_{QDF}(x) = -\frac{1}{2}(x - \bar{x}_0)^T \underbrace{(\alpha \times I_p + C_0)^{-1}}_{K_0} (x - \bar{x}_0) + \frac{1}{2}(x - \bar{x}_1)^T \underbrace{(\alpha \times I_p + C_1)^{-1}}_{K_1} (x - \bar{x}_1) + \frac{1}{2}\log\left(\frac{|\alpha \times I_p + C_1|}{|\alpha \times I_p + C_0|}\right)$$

where $\alpha \ge 0.1$ is regularization constant, I_p is identity matrix.

$\bar{x}_0 = \frac{1}{n_0}\sum_{x_l \in S_0^n} x_l$ and $\bar{x}_1 = \frac{1}{n_1}\sum_{x_l \in S_1^n} x_l$ are sample means of 0 and 1 classes. C_i is covariance matrix of each class where $i = \{0,1\}$.

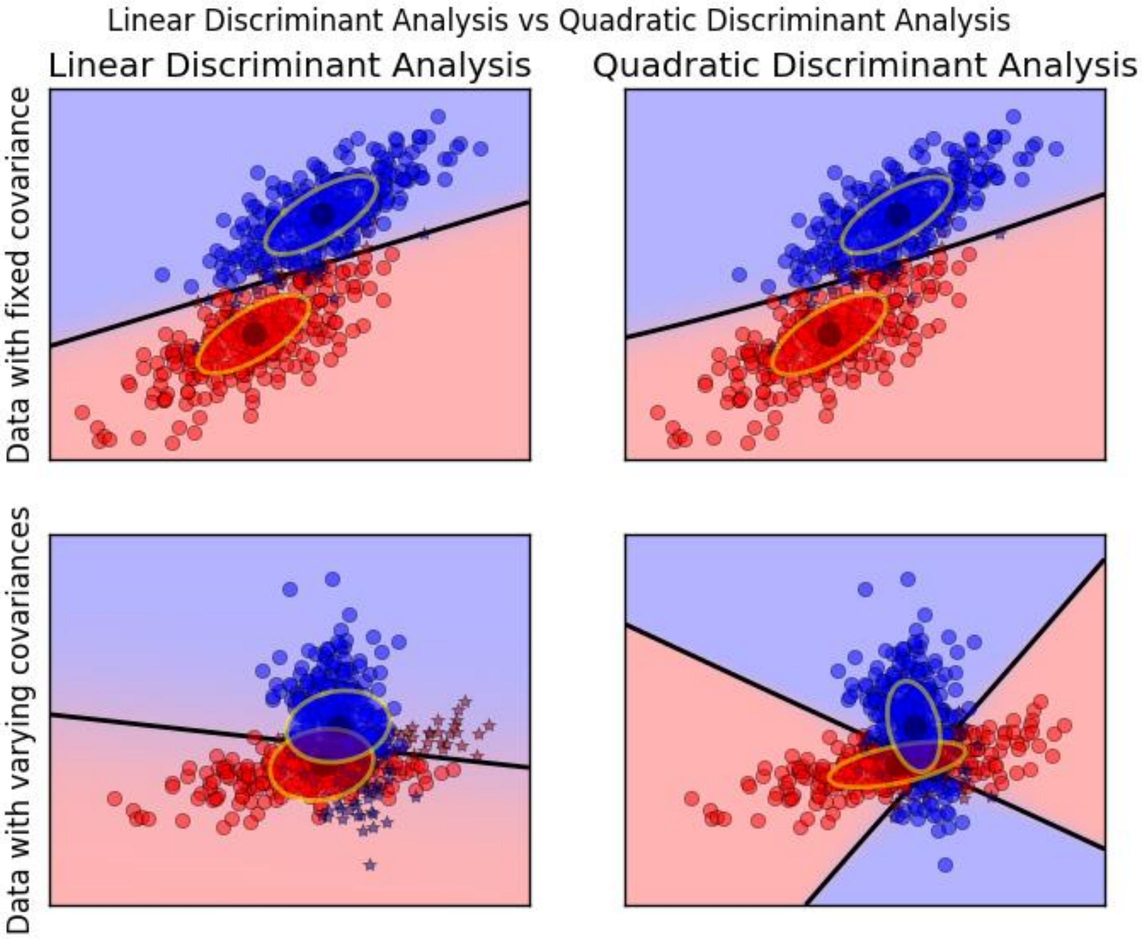
$$C_i = \frac{1}{n_i - 1}\sum_{x_l \in S_i^n} (x_l - \bar{x}_i)\,(x_l - \bar{x}_i)^T$$

The QDA classifier is

$$\psi_{QDA}(x) = \begin{cases} 1, & if\ W_{QDF}(x) \le c \\ 0, & if\ W_{QDF}(x) > c \end{cases}$$

where $c = \log\frac{N_1}{N_0}$

The decision boundary is curved if the covariance matrices are different. So, when considering two classes QDA is usually more powerful than LDA in predicting the classes. And if covariance matrices of two classes are same, then QDA works as LDA, and decision boundary is linear. It is best illustrated in the graph below.



Graph 2. Illustration of separation of QDA and LDA (ScikitLearn n.d).

Both LDA and QDA classifiers perform well various classification tasks. For example, in the STATLOG project LDA classifier was among the top three classifiers in 7 datasets among the 22 datasets, QDA was among the top three in 4 datasets, and one of them were in the top three among 10 datasets (Trevor, Hastie and Jerome).

3.2.3. Diagonal Linear Discriminant Analysis

The DLDA is modified version of LDA, but the only difference is that in DLDA the pooled covariance matrix was replaced by D matrix, where diagonal elements remains the same as covariance matrix, but non-diagonal elements are zero. DLDA works for any sample size(N) and number of features(d), because D matrix is always invertible due to all elements of D matrix are positive. Despite d does not depend on N, according to Pang, Herbert and Michael it often fails in classifying. But, according to Keith, Baggerly and Kevin, 2004 DLDA is much better behaved for microarray data and has been successfully employed for high-dimensional problems. So, we analyze it. DLDA is implemented using the following formula:

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

where $\bar{x}_0 = \frac{1}{n_0}\sum_{x_l \in S_0^n} x_l$ and $\bar{x}_1 = \frac{1}{n_1}\sum_{x_l \in S_1^n} x_l$ are sample means of 0 and 1 classes. D is the matrix, where diagonal elements are same as C pooled sample covariance matrix and non-diagonal elements are 0.

The DLDA classifier is

$$\psi_{DLDA}(x) = \begin{cases} 1, & if\ W_{LDF}(x) \le c \\ 0, & if\ W_{LDF}(x) > c \end{cases}$$

where $c = \log\frac{N_1}{N_0}$.

3.2.4. G13

G13 is also modified version of LDA, but only the value of the discriminant function is multiplied by $\left(\frac{n_0+n_1-2-p}{n_0+n_1-2}\right)$. It is implemented by the following formula:

$$W_{G13}(x) = \left(\left(x - \frac{\bar{x}_0 + \bar{x}_1}{2}\right)^T C^{-1}(\bar{x}_0 - \bar{x}_1)\right)\left(\frac{n_0+n_1 - 2 - p}{n_0 + n_1 - 2}\right)$$

where $\bar{x}_0 = \frac{1}{n_0}\sum_{x_l \in S_0^n} x_l$ and $\bar{x}_1 = \frac{1}{n_1}\sum_{x_l \in S_1^n} x_l$ are sample means of 0 and 1 classes. C is pooled sample covariance matrix

$$C = \frac{(n_0 - 1)C_0 + (n_1 - 1)C_1}{n_0 + n_1 - 2}$$

where C_i is covariance matrix of each class where $i = \{0,1\}$

$$C_i = \frac{1}{n_i - 1}\sum_{x_l \in S_i^n} (x_l - \bar{x}_i)\,(x_l - \bar{x}_i)^T$$

The G13 classifier is

$$\psi_{G13}(x) = \begin{cases} 1, & if\ W_{G13}(x) \le c \\ 0, & if\ W_{G13}(x) > c \end{cases}$$

3.2.5. RLDA

As LDA requires p<N due to singularity when p≥N, it needs to regularize its covariance matrix to use it for p≥N. RLDA is regularized version of LDA and it shrinks towards its

diagonal. RLDA is initially designed for data that has high dimension and low sample size. According to Guo et al RLDA is competitive classifier as SVM classifiers. RLDA is implemented:

$$W_{RLDF}(x) = \left(x - \frac{\bar{x}_0 + \bar{x}_1}{2}\right)^T (\gamma \times I_p + C)^{-1}(\bar{x}_0 - \bar{x}_1)$$

The RLDA classifier is

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

The main idea is that, using the training data, that you already have, to find the optimal γ for building the classifier for next predictions. To do so, training data with N sample size is divided into two sub-sets S$_1$ and S$_2$ with [2N/3] and N-[2N/3] sample size respectively, where [a] denotes the maximum integer number less than a. Then using S$_1$ and S$_2$ as training data and tst data respectively, and by changing the γ each time the errors will be calculated. Then the optimal γ, that corresponds to minimum error among all errors, will be used to build the RLDA for next predictions.

3.2.6. SEDC

(NOTE: we have multiplied the discriminant function by m, in our case SEDC was built for k=1, which means m=p)

SEDC is the modified form of the simple Linear Discriminant Analyze. It has an advantage of efficiency in compare with Linear Discriminant Analyze when there are high dimensional matrix with data. There is a dependence of discrimination error probabilities on the dimension and sample size. In SEDC the dimensions of vectors are reduced by k from d. This helps to keep the efficiency of LDA with high dimensional matrices and it is referred as SEDC (Serdobolskii 2010). SEDC is implemented by following formula:

If $p = mk$ such that $p > k \geq 1$ and $m > 1$ and are positive integers and

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_p \end{bmatrix} \rightarrow z = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m} x_i \\ \frac{1}{m}\sum_{i=m+1}^{2m} x_i \\ \frac{1}{m}\sum_{i=2m+1}^{3m} x_i \\ \vdots \\ \frac{1}{m}\sum_{i=(k-1)m+1}^{km} x_i \end{bmatrix}$$

$$W_{SEDC}(x) = m\left(z - \frac{\bar{z}_0 + \bar{z}_1}{2}\right)^T (\bar{z}_0 - \bar{z}_1)$$

The SEDC classifier is

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

where $c = \log \frac{N_1}{N_0}$.

3.2.7. SVM - Support Vector Machine.

Support vector machine is a supervised learning method related to pattern recognition and data analysis. It writes an algorithm that gives optimal hyperplane for defining to which category new data belongs. There are two types of SVM: Linear SVM, Kernel SVM (OpenCV n.d.).

LSVM

Linear support vector machine separates data set of points yi=1 from yi=-1, where yi denotes the class labels, i=1,...n and takes the maximum gap between hyperplane and support vectors. The hyperplane is placed between two support vectors and the distance between two support vectors is called margin. The distance between two support vectors is 2/||w||. The larger size of margin the better classification. To maximize margin, we need to minimize ||w|| (Village, Berwick n.d.).

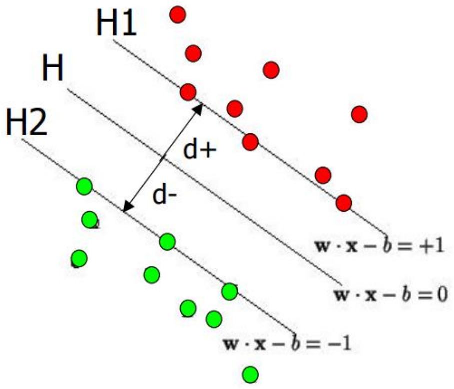

H1
H
H2
d+
d-
$\mathbf{w} \cdot \mathbf{x} - b = +1$
$\mathbf{w} \cdot \mathbf{x} - b = 0$
$\mathbf{w} \cdot \mathbf{x} - b = -1$

Figure 1: Classifying the objects by LSVM

KSVM

Kernel SVM is used when data cannot be separated linearly. Kernel methods can operate in high-dimensional space with difficult distinguishable features, by computing inner products between the images with different type of data. For example, people use Kernel SVM for classifying images. It defines a single function that compute similarity between images, instead of defining many features of images (Jason n.d.).

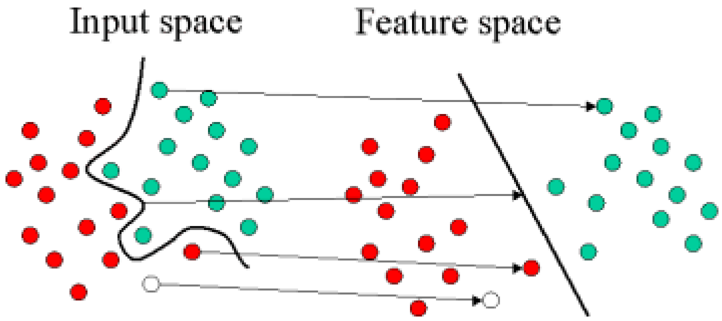
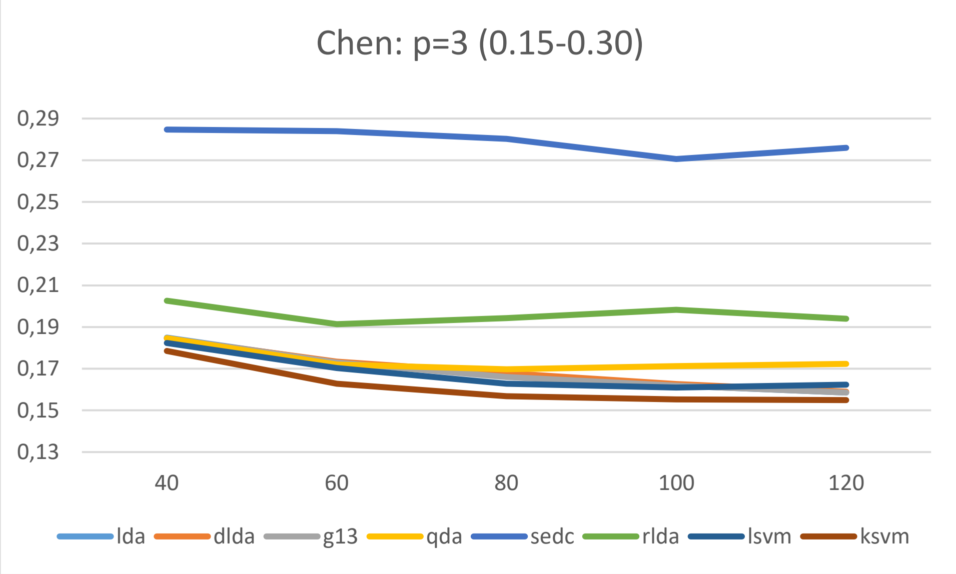


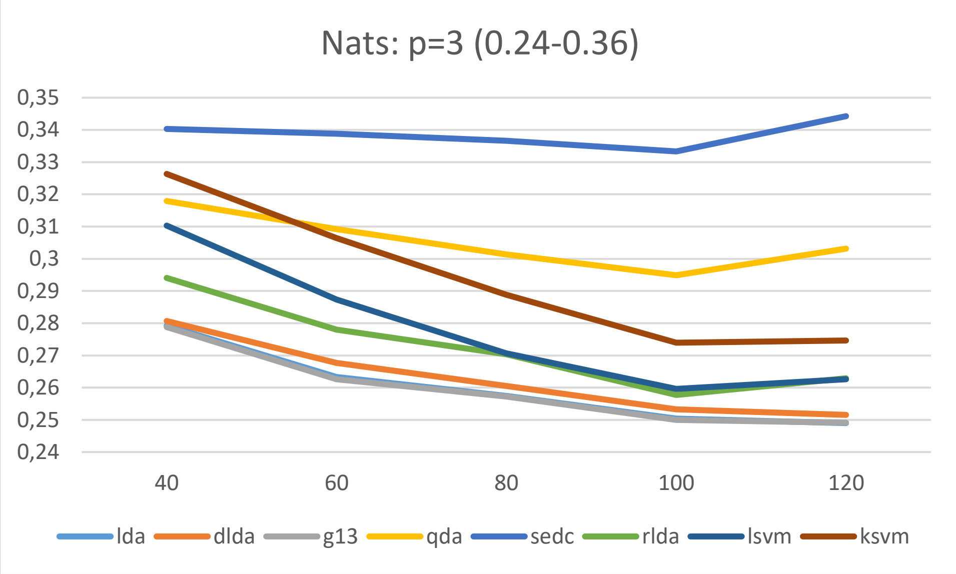
Figure 2: Separating the classes into two groups by KSVM

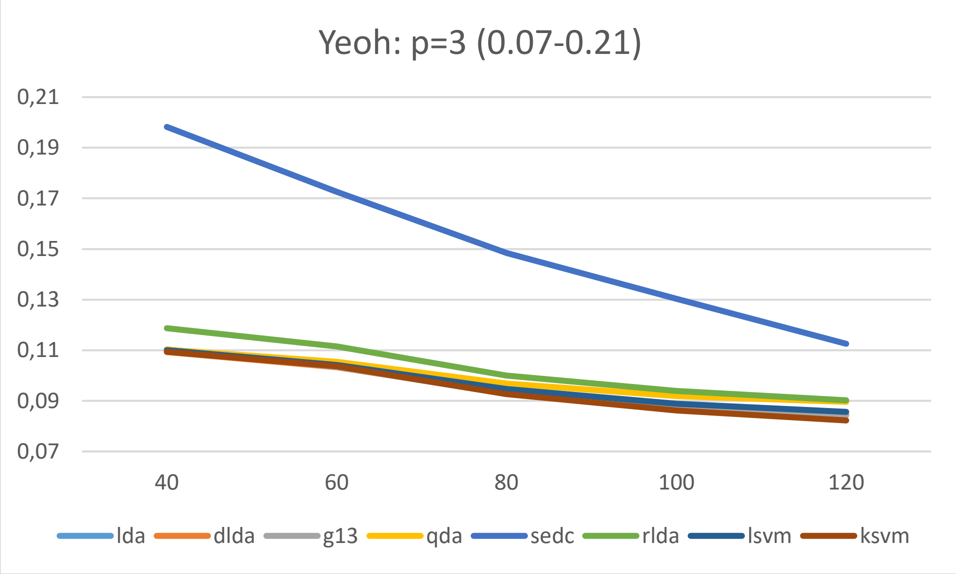
NOTE: To find the true error of each classifier, any object from the training data must not be included in tst data, otherwise the obtained error is not the true error of the classifier.

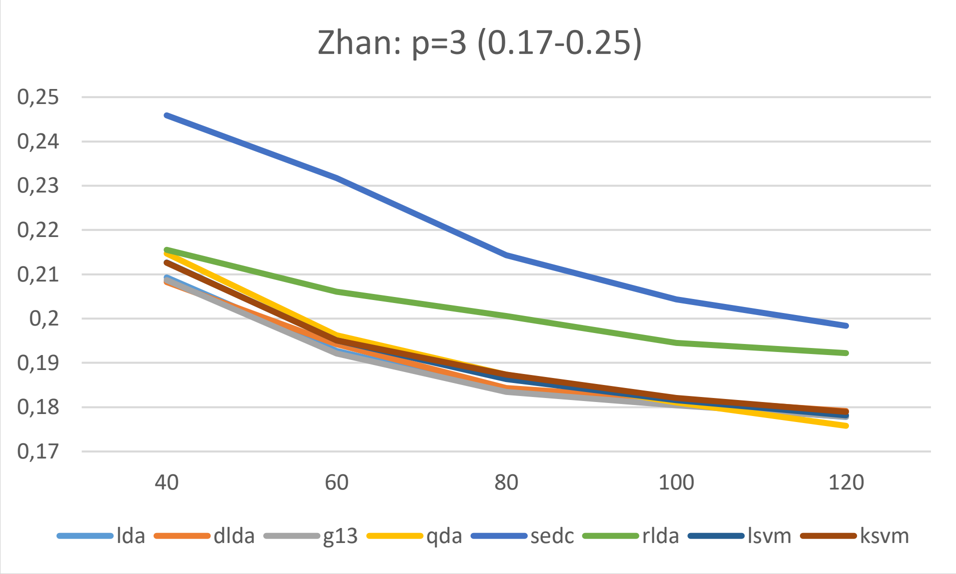
4. RESULTS AND DISCUSSION

p=3

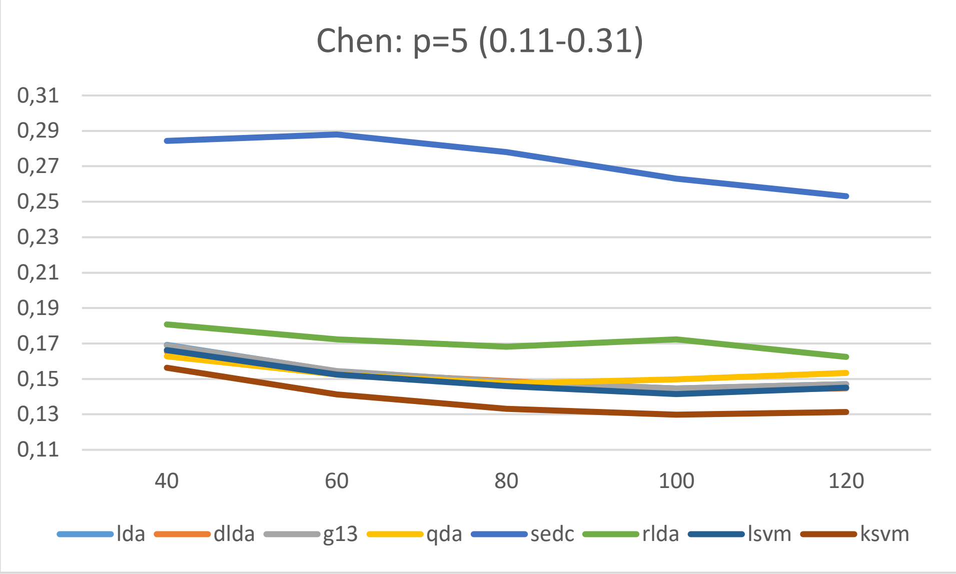

Chen: p=3 (0.15-0.30)

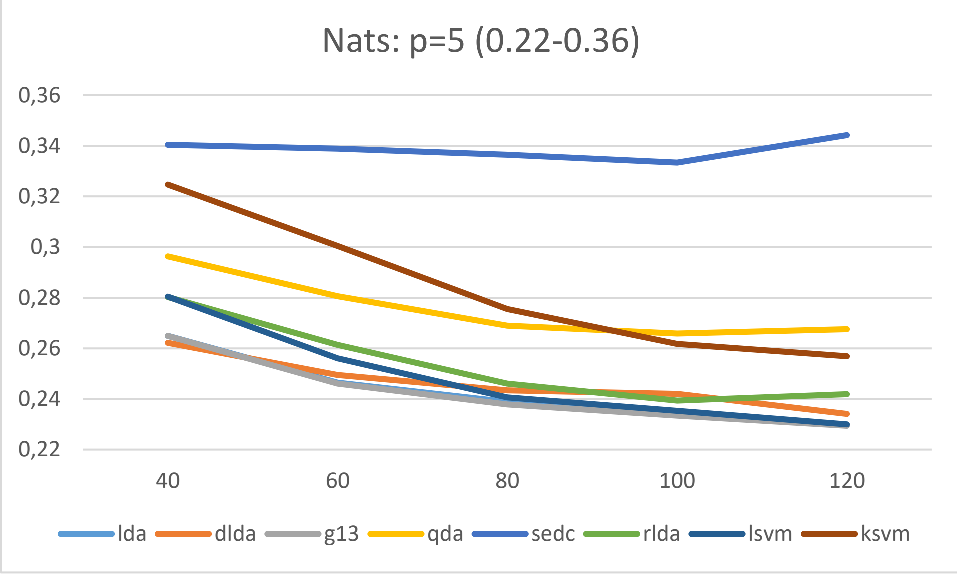

Nats: p=3 (0.24-0.36)

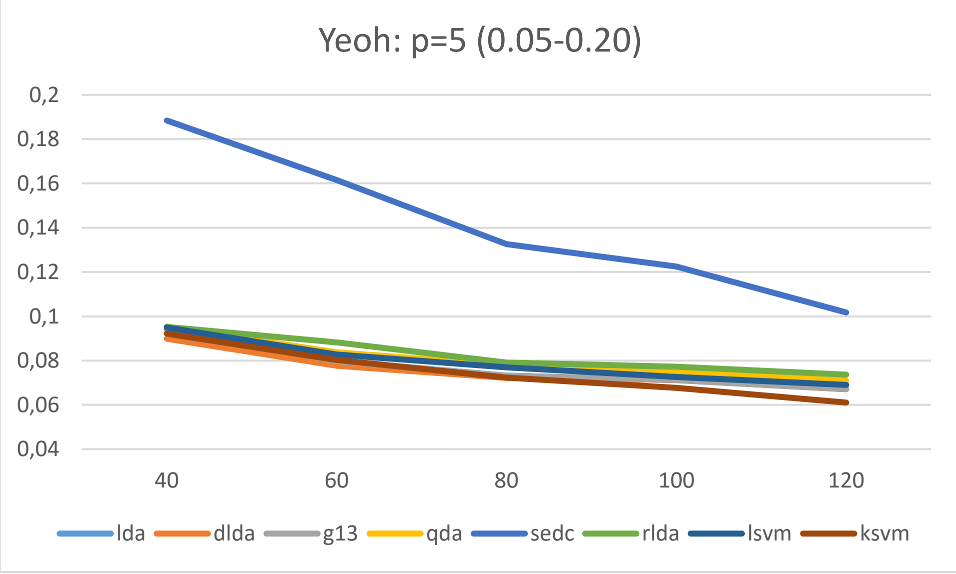

Yeoh: p=3 (0.07-0.21)

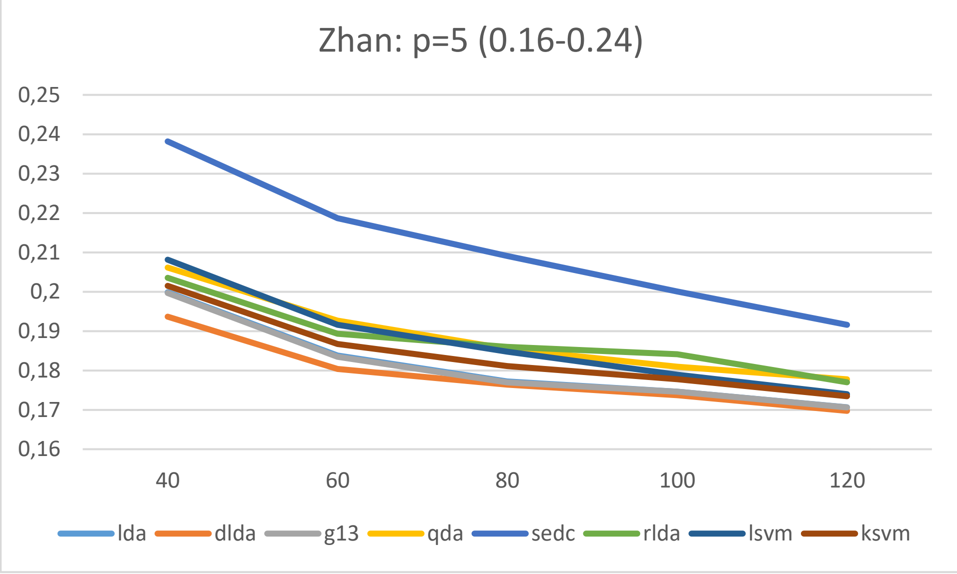

Zhan: p=3 (0.17-0.25)

p=5

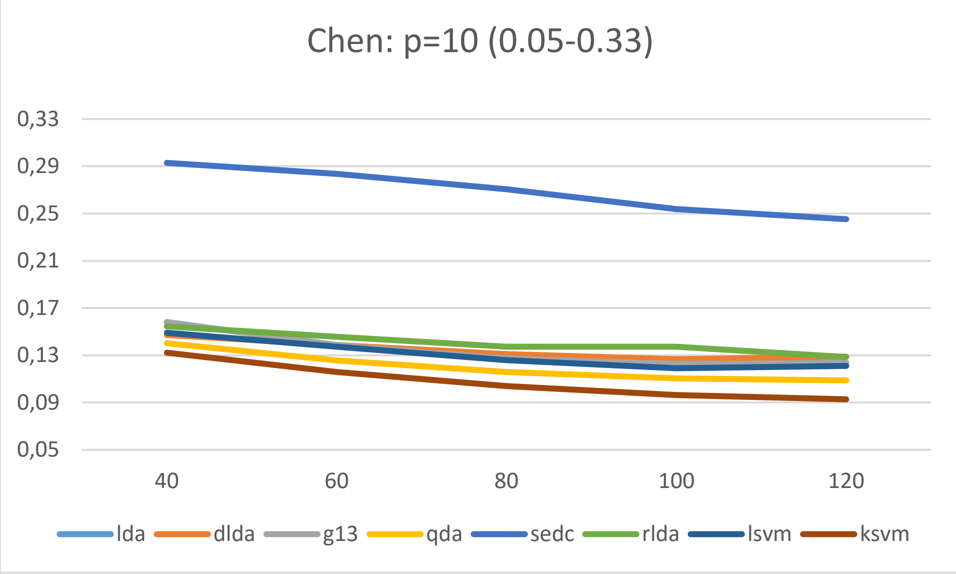

Chen: p=5 (0.11-0.31)
0,31
0,29
0,27
0,25
0,23
0,21
0,19
0,17
0,15
0,13
0,11
40
60
80
100
120
lda
dlda
g13
qda
sedc
rlda
lsvm
ksvm

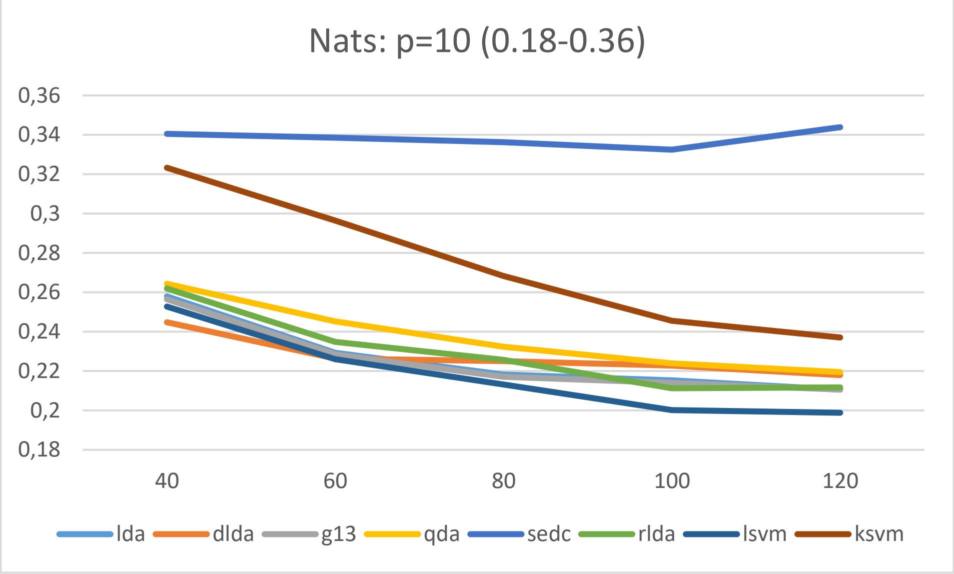

Nats: p=5 (0.22-0.36)
0,36
0,34
0,32
0,3
0,28
0,26
0,24
0,22
40
60
80
100
120
lda
dlda
g13
qda
sedc
rlda
lsvm
ksvm

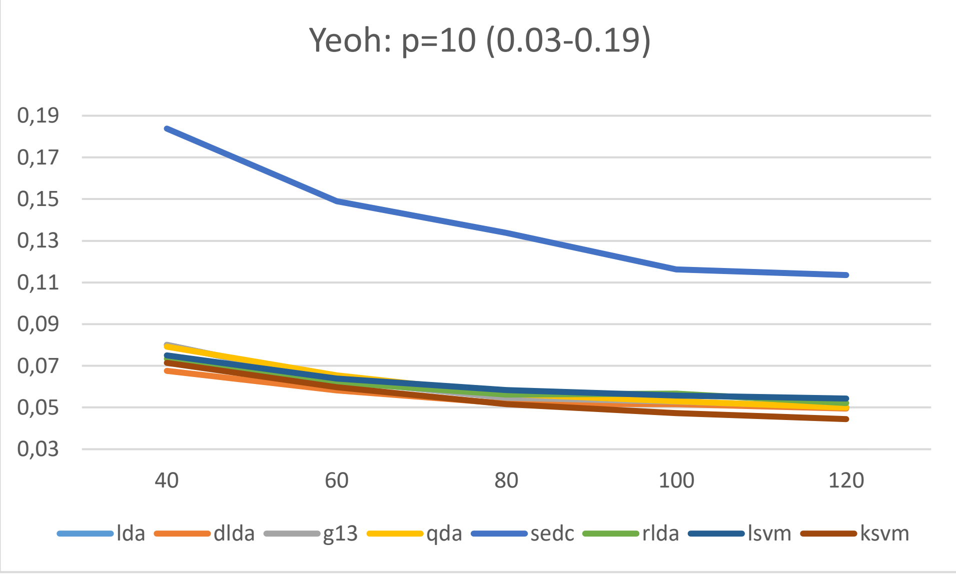

Yeoh: p=5 (0.05-0.20)
0,2
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
40
60
80
100
120
lda
dlda
g13
qda
sedc
rlda
lsvm
ksvm

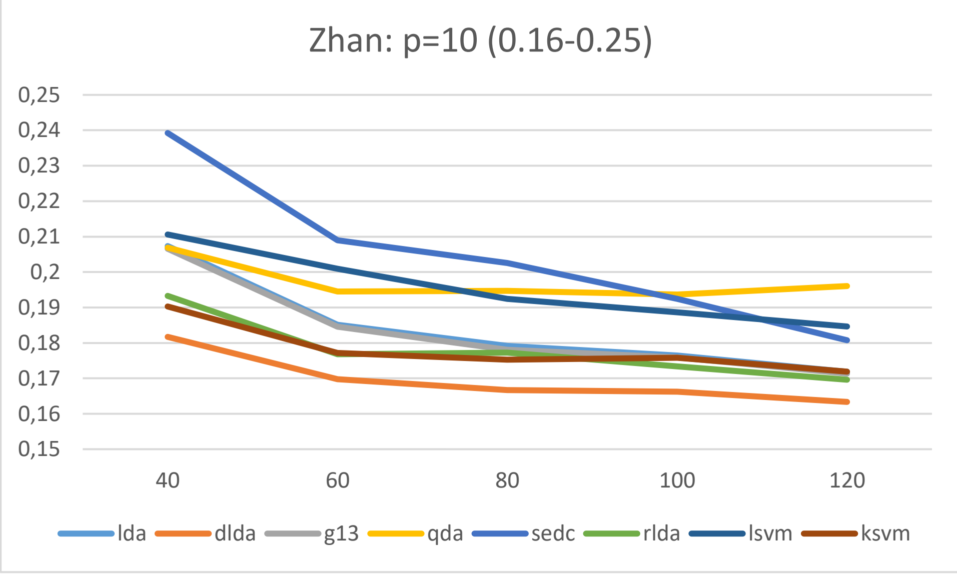

Zhan: p=5 (0.16-0.24)
0,25
0,24
0,23
0,22
0,21
0,2
0,19
0,18
0,17
0,16
40
60
80
100
120
lda
dlda
g13
qda
sedc
rlda
lsvm
ksvm

p=10

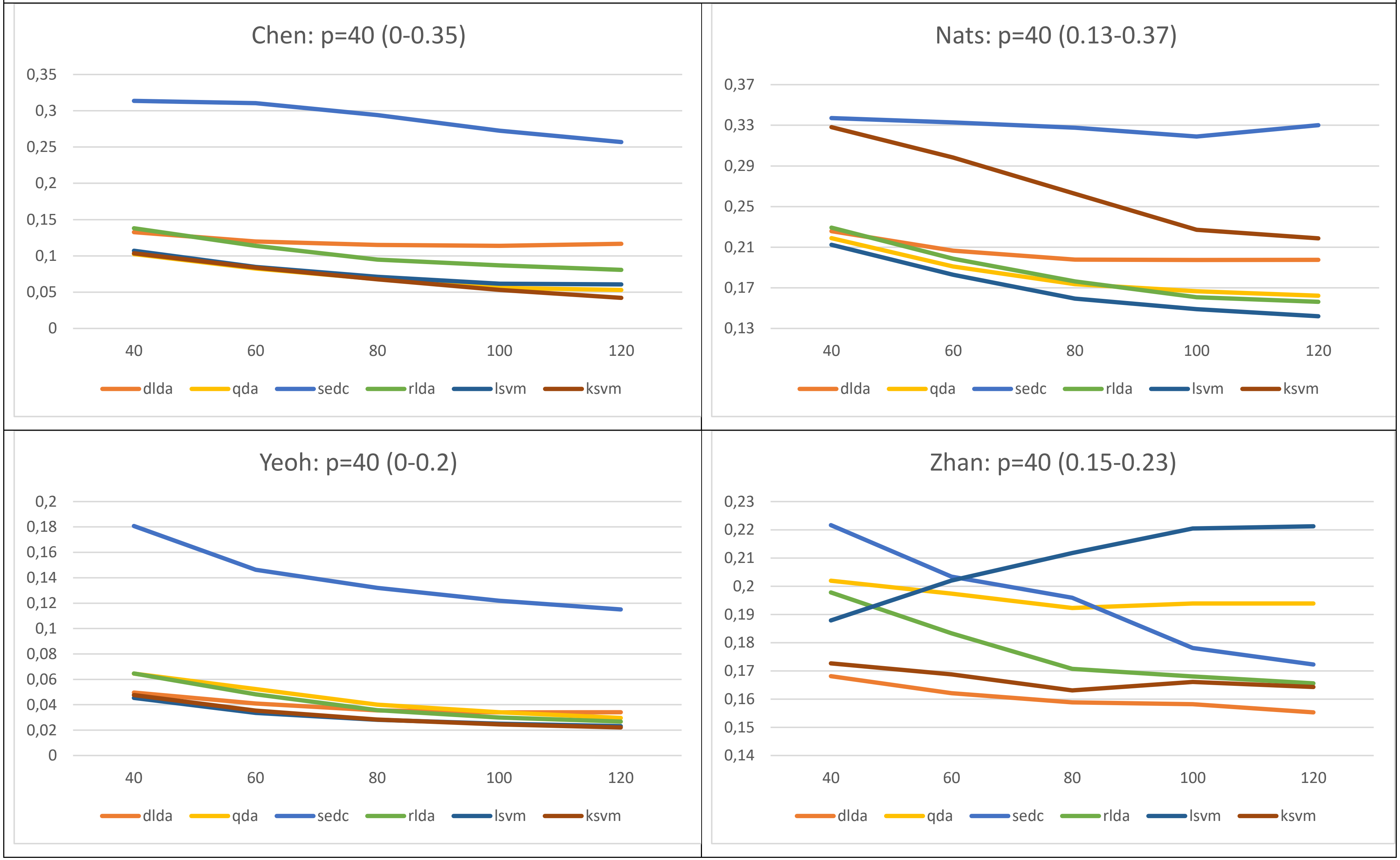

Chen: p=10 (0.05-0.33)
0,33
0,29
0,25
0,21
0,17
0,13
0,09
0,05
40
60
80
100
120
lda
dlda
g13
qda
sedc
rlda
lsvm
ksvm


Nats: p=10 (0.18-0.36)
0,36
0,34
0,32
0,3
0,28
0,26
0,24
0,22
0,2
0,18
40
60
80
100
120
lda
dlda
g13
qda
sedc
rlda
lsvm
ksvm


Yeoh: p=10 (0.03-0.19)
0,19
0,17
0,15
0,13
0,11
0,09
0,07
0,05
0,03
40
60
80
100
120
lda
dlda
g13
qda
sedc
rlda
lsvm
ksvm

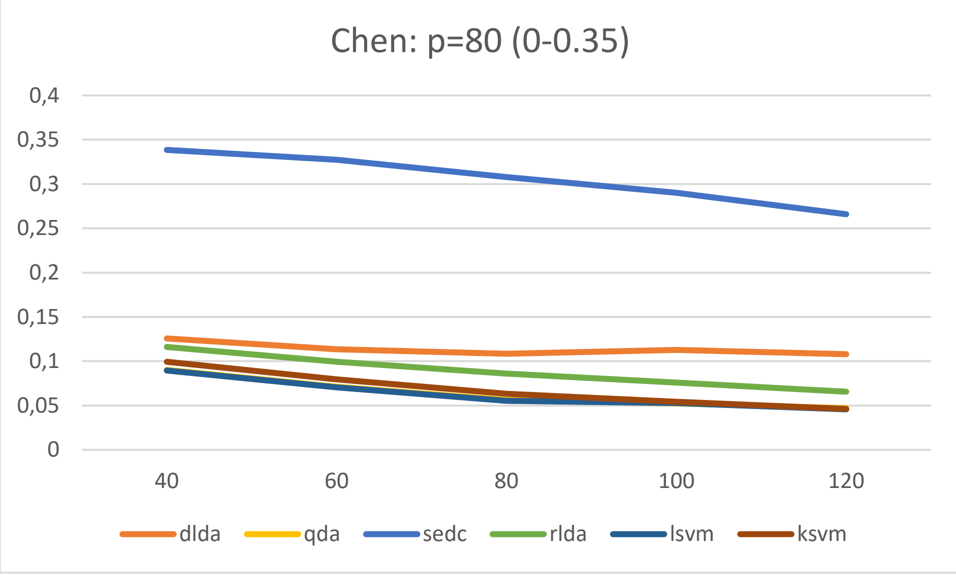

Zhan: p=10 (0.16-0.25)
0,25
0,24
0,23
0,22
0,21
0,2
0,19
0,18
0,17
0,16
0,15
40
60
80
100
120
lda
dlda
g13
qda
sedc
rlda
lsvm
ksvm

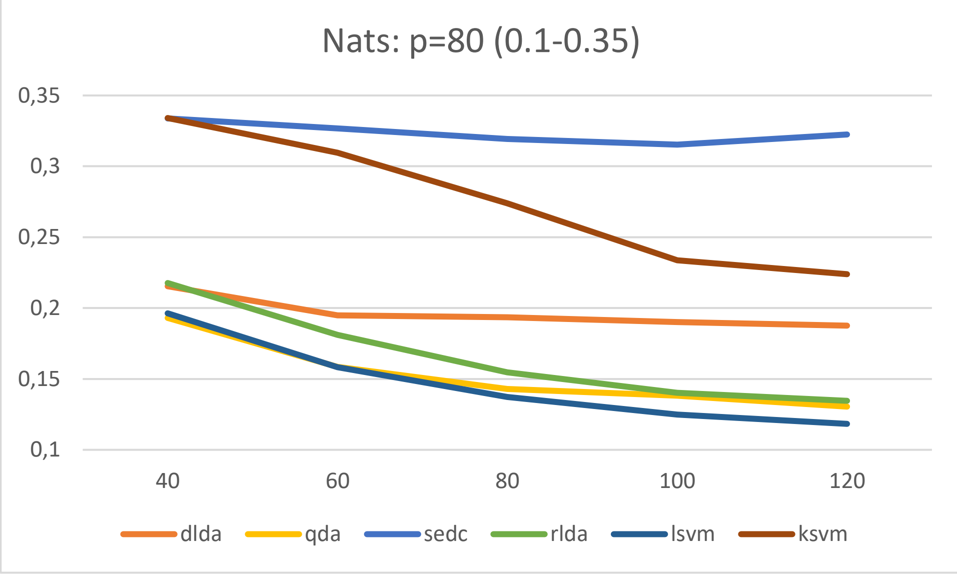
p=40

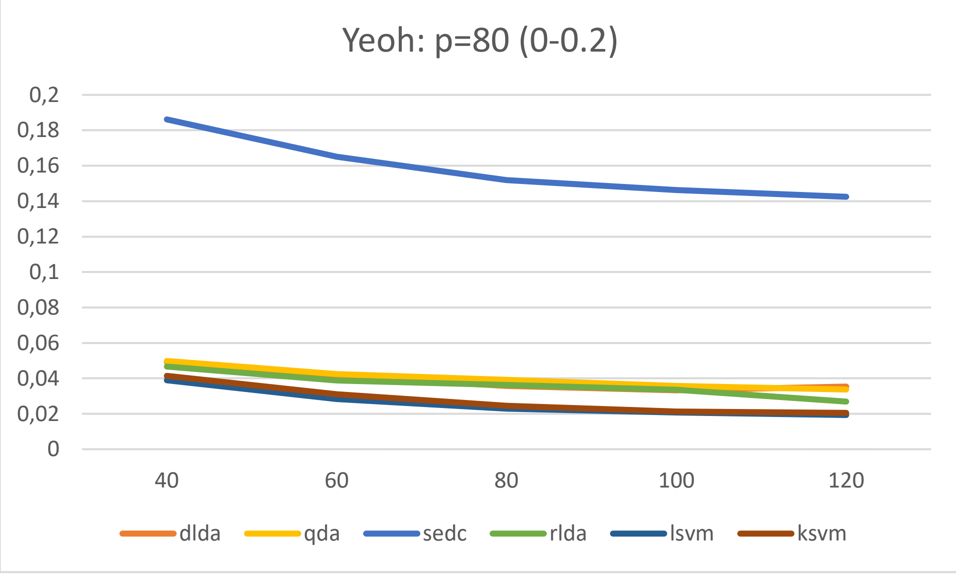


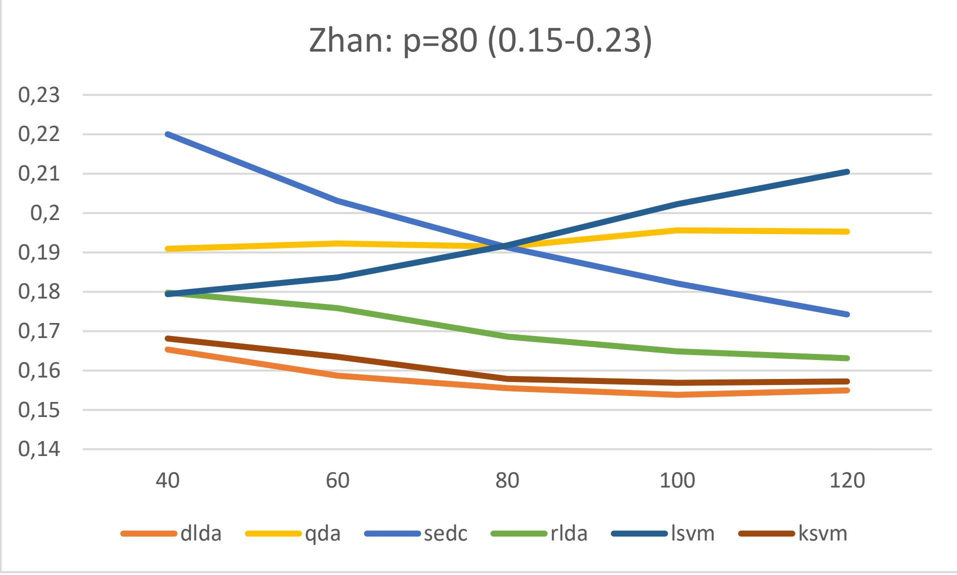
Chen: p=40 (0-0.35)
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
40
60
80
100
120
dlda
qda
sedc
rlda
lsvm
ksvm
Nats: p=40 (0.13-0.37)
0,37
0,33
0,29
0,25
0,21
0,17
0,13
40
60
80
100
120
dlda
qda
sedc
rlda
lsvm
ksvm
Yeoh: p=40 (0-0.2)
0,2
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
40
60
80
100
120
dlda
qda
sedc
rlda
lsvm
ksvm
Zhan: p=40 (0.15-0.23)
0,23
0,22
0,21
0,2
0,19
0,18
0,17
0,16
0,15
0,14
40
60
80
100
120
dlda
qda
sedc
rlda
lsvm
ksvm

p=80


Chen: p=80 (0-0.35)
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
40
60
80
100
120
dlda
qda
sedc
rlda
lsvm
ksvm


Nats: p=80 (0.1-0.35)
0,35
0,3
0,25
0,2
0,15
0,1
40
60
80
100
120
dlda
qda
sedc
rlda
lsvm
ksvm


Yeoh: p=80 (0-0.2)
0,2
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
40
60
80
100
120
dlda
qda
sedc
rlda
lsvm
ksvm


Zhan: p=80 (0.15-0.23)
0,23
0,22
0,21
0,2
0,19
0,18
0,17
0,16
0,15
0,14
40
60
80
100
120
dlda
qda
sedc
rlda
lsvm
ksvm

To analyze the classifiers' true error and find best one the output results were analyzed separately. Firstly, the output results from 3, 5 and 10 features were analyzed, because the number of features is much less than 40 and 80 features. 40 and 80 features were analyzed secondly.

From the graphs, it is clearly seen that, as dimension and sample size increases then the true error of classifiers decreases. Lower true error means better performance. Therefore, it is clearly seen that, SEDC fails to classify the objects on almost all dimensions and data, and its true error is about two times higher than other classifiers which have the true error at most 0.3.

When $3 \leq p \leq 10$, the classifiers RLDA and QDA show the medium true error while the best classifiers such as KSVM, DLDA, G13 and LSVM share the 3 places. The KSVM shows the best results while DLDA with G13 show the same medium result and LSVM shows the good results. Since both SVM members KSVM and LSVM are on the podium, it can be seen that SVM classifiers are more accurate when $p \leq 10$. The classifiers G13 and LDA cannot classify the object when $p \geq 40$.

When $40 \leq p \leq 80$, there were compared RLDA, QDA, DLDA, SCEDC, KSVM and LSVM. As the graphs showed the output true error of RLDA and QDA was medium. While the KSVM and LSVM showed that they are more accurate than QDA and RLDA. However, KSVM and LSVM have a rapid change of error amount. For example, the KSVM loosed its accuracy when it classified data of Nats: $p=80$. The LSVM showed low accuracy when it classified the data of Zhan: $p=80$. In other hand, DLDA showed the lowest accuracy in all data, except Zhan: $p=80$. It has the most accurate classification when it classified the data of Zhan. So then, our analysis showed that, it is difficult to determine the most accurate classifier.

5. CONCLUSION

Our analysis showed that classifiers' accuracy can be varied due to the different data. For example, to classify the data Zhan: $p=80$ DLDA will be the most appropriate classifier, but DLDA shows one of the lowest accuracy for other three data. Therefore, determining the best classifier is difficult. If we chose the classifier that, its true error does not fluctuate (stable) due to the different data, the best classifier is RLDA, because it is the most accurate classifier after the KSVM, LSVM and DLDA. Moreover, as researcher Guo(as mentioned in the system and methods) said that the RLDA can achieve the accuracy of SVM family if the regularization process of RLDA is well improved. For example, in this project γ opt was chosen among the 21 real numbers between 0.001 and 0.001×2^{20} with geometrical progress, if it was chosen between 0.0001 and $0.0001\times(1.2)^{100}$ it would be well regularized. It proves that RLDA is the best classifier. If Discriminant analysis family and SVM family are compared, as the result of first and second analyze showed the SVM family is more accurate than Discriminant family. In conclusion, the graphs showed that the best classifier is almost RLDA and the best family is SVM family.

#	File names	Average of all numbers
1.	ChenLiver	-0.03187
2.	Natsoulis	0.003788
3.	YeohLeukemia	12.07733
4.	ZhanMyeloma	7.65311

P.S. the effect of multiplication by **m** is significant when we consider the Natsoulis.txt. After the multiplication the true error of SEDC is reduced about 2 times.

3.3. CODING THE CLASSIFIERS:

All mathematical operations was explained using matrix identities.

Removing redundant codes (for LEVEL 1)

#	Source file	Removed lines from original project (not deleted, commented)
1.	calculateErrors.h	24-42 lines
2.	calculateErrors.cpp	22-25, 46-60, 82-100 lines
3.	errorEstimation.h	Exception (it can be left or removed): 16-30 lines
4.	errorEstimation.cpp	Exception (it can be left or removed): 8-1037 lines
5.	main.cpp	51-69, 81-99, 192-296, 316-334, 342-360, 379-393 lines

3.3.1. LDA

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

$$= \underbrace{x^T \underbrace{C^{-1}(\bar{x}_1 - \bar{x}_0)}_{lda.a} \underbrace{-0.5(\bar{x}_0 + \bar{x}_1)^T \underbrace{C^{-1}(\bar{x}_1 - \bar{x}_0)}_{lda.a} + c}_{lda.b}}_{the\ code\ was\ written\ according\ to\ this}$$

3.3.2. DLDA

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

$$= \underbrace{x^T \underbrace{D^{-1}(\bar{x}_1 - \bar{x}_0)}_{dlda.a} \underbrace{-0.5(\bar{x}_0 + \bar{x}_1)^T \underbrace{D^{-1}(\bar{x}_1 - \bar{x}_0)}_{dlda.a} + c}_{dlda.b}}_{the\ code\ was\ written\ according\ to\ this}$$

Source files:

classifier.h (dlda):

1. It declares the struct DLDA with `double *a,` which holds vector, and with `double b,` which holds scalar constant value of $c - W_{DLDF}(x)$ indicated as *dlda.b.*

2. Prototype of functions: void dldaTrn, double dldaTst.

```
#  Codes
1. struct DLDA {
       double *a;
       double b;
   };
2. void dldaTrn(double** X, int* y, int N, int d, int* ind, DLDA* dlda);
   double dldaTst(double** X, int* y, int N, int d, int* ind, DLDA dlda);
```

classifier.cpp (dlda)

To make dlda.Trn and dldaTst, code of ldaTrn and ldaTst were copied and pasted after ldaTst. Then all keywords, that indicate lda and LDA were changed to dlda and DLDA. Then only following code was added to do so, pooled sample covariance matrix become D (with 0 off-diagonal elements). The code added after the line, where pooled_cov is calculated.

1. If i≠j, it means that, it is the indexes of non-diagonal elements. So, the following code initializes values of non-diagonal elements to 0 as we desired.

```
#   Codes
1.  for(i=0; i<d; i++) {
        for(j=0; j<d; j++) {
            if(i!=j)
                pooled_cov[i][j] = 0;
        }
    }
```

calculateErrors.h (dlda)

1. `dlda_true_error` is declared as double and it holds the value of the true error of DLDA.

```
#   Codes
1.  double dlda_true_error;
```

calculateErrors.cpp (dlda)

1. It declares `model_DLDA` as variable of struct `DLDA`.

2. The `(*all_errors).dlda_true_error` is initialized to `0`.

3. The size of `model_DLDA.a array` is initialized to `d`, and it can be deleted from the memory because of it is **`new double`**.

4. `dldaTrn` function is invoked, and main purpose of `dldaTrn` is to calculate the value of `model_DLDA.a` and `model_DLDA.b`

5. `(*all_errors).dlda_true_error` takes the return value of `dldaTst` function.

6. `model_DLDA.a` array is deleted from the memory.

```
#   Codes
1.  DLDA model_DLDA;
2.  (*all_errors).dlda_true_error = 0.00;
3.  model_DLDA.a = new double [d];
```

4.	dldaTrn(data_trn.data, data_trn.labels, data_trn.N, d, best_features, &model_DLDA);
5.	(*all_errors).dlda_true_error = dldaTst(data_tst.data, data_tst.labels, data_tst.N, d, best_features, model_DLDA);
6.	delete model_DLDA.a;

main.cpp (dlda)

1. The value of `double dlda_true_error` after initiliztion will be written in output .txt file.

2. Declaration of `FILE* fp_dlda_true_error;`

3. It makes the file with the name `*_dlda_true_error_round=1.txt,` where `*` is given by the command line.

4. `dlda_true_error` is initialized.

5. The value of `dlda_true_error` is written in `*_dlda_true_error_round=1.txt` file.

6. File is closed.

#	Codes
1.	double dlda_true_error;
2.	FILE* fp_dlda_true_error;

3.	sprintf(outputFilenameFinal, "%s_dlda_true_error_round=%d.txt", outputFilename, round);
	fp_dlda_true_error = fopen(outputFilenameFinal, "w");
	if(fp_dlda_true_error==NULL)
	{
	    printf("cannot open %s_dlda_true_error_round=%d.txt", outputFilename, round);
	    return(-1);
	}

4.	dlda_true_error = all_errors.dlda_true_error;
5.	fprintf(fp_dlda_true_error, "%f\n", dlda_true_error);
6.	fclose(fp_dlda_true_error);

3.3.3. G13

The code of G13 was written according to this equation:

$$c - W_{G13}(x) = c - \left(\left(x - \frac{\bar{x}_0 + \bar{x}_1}{2}\right)^T C^{-1}(\bar{x}_0 - \bar{x}_1)\right)\left(\frac{n_0 + n_1 - 2 - p}{n_0 + n_1 - 2}\right)$$

$$= \underbrace{x^T \underbrace{C^{-1}(\bar{x}_1 - \bar{x}_0)\left(\frac{n_0 + n_1 - 2 - p}{n_0 + n_1 - 2}\right)}_{g13.a} \underbrace{-0.5(\bar{x}_0 + \bar{x}_1)^T \underbrace{C^{-1}(\bar{x}_1 - \bar{x}_0)\left(\frac{n_0 + n_1 - 2 - p}{n_0 + n_1 - 2}\right)}_{g13.a}}_{g13.b} + c}_{\textit{the code was written according to this}}$$

classifier.h (g13):

1. It declares the struct G13 with `double *a`, which holds vector, and with `double b`, which holds scalar constant value of $c - W_{G13}(x)$ indicated as *g13.b.*

2. Prototype of functions: void g13Trn, double g13Tst.

#	Codes

1.	`struct G13 {`
	`    double *a;`
	`    double b;`
	`};`

2.	`void G13Trn(double** X, int* y, int N, int d, int* ind, G13* g13);`
	`double g13Tst(double** X, int* y, int N, int d, int* ind, G13 g13);`

classifier.cpp (g13)

To make g13Trn and g13Tst, code of ldaTrn and ldaTst were copied. Then all keywords, that indicate lda and LDA were changed to g13 and G13. Then only following code was added to do so, every element of $(\bar{x}_0 - \bar{x}_1)$ is multiplied by $\left(\frac{n_0+n_1-2-p}{n_0+n_1-2}\right)$.

1. multiplier = $1/\left(\frac{n_0+n_1-2-p}{n_0+n_1-2}\right)$. The multiplier in `sum_two_matrix` function divides the result. All in all, it multiplies $(\bar{x}_0 - \bar{x}_1)$ by $\left(\frac{n_0+n_1-2-p}{n_0+n_1-2}\right)$.

#	Codes

1.	`multiplier = ((double)(N_0 + N_1 -2)) / ((double)(N_0 + N_1 - 2 - d));`
	
	`sum_two_matrix(&mu_hat_1, &mu_hat_0, 1, d, 1.00, -1.00, multiplier,`
	`&temp);`

calculateErrors.h (g13)

1. `g13_true_error` is declared as double and it holds the value of the true error of G13.

#	Codes
1.	`double g13_true_error;`

calculateErrors.cpp (g13)

1. It declares `model_G13` as variable of struct `G13`.

2. The `(*all_errors).g13_true_error` is initialized to `0`.

3. The size of `model_G13.a array` is initialized to `d`, and it can be deleted from the memory because of it is `new double`.

4. g13Trn function is invoked, and main purpose of g13Trn is to calculate the value of model_G13.a and model_G13.b

5. (*all_errors).g13_true_error takes the return value of g13Tst function.

6. model_G13.a array is deleted from the memory.

#	Codes
1.	G13 model_G13;
2.	(*all_errors).g13_true_error = 0.00;
3.	model_G13.a = **new** **double** [d];
4.	g13Trn(data_trn.data, data_trn.labels, data_trn.N, d, best_features, &model_G13);
5.	(*all_errors).g13_true_error = g13Tst(data_tst.data, data_tst.labels, data_tst.N, d, best_features, model_G13);
6.	**delete** model_G13.a;

main.cpp (g13)

1. The value of **double** g13_true_error after initiliztion will be written in output .txt file.

2. Declaration of FILE* fp_g13_true_error;

3. It makes the file with the name *_g13_true_error_round=1.txt, where * is given by the command line.

4. g13_true_error is initialized.

5. The value of g13_true_error is written in *_g13_true_error_round=1.txt file.

6. File is closed.

#	Codes
1.	**double** g13_true_error;
2.	FILE* fp_g13_true_error;

3.	sprintf(outputFilenameFinal, "%s_g13_true_error_round=%d.txt", outputFilename, round);
	fp_g13_true_error = fopen(outputFilenameFinal, "w");
	if(fp_g13_true_error==NULL)
	{
	    printf("cannot open %s_g13_true_error_round=%d.txt", outputFilename, round);
	    return(-1);
	}

4.	g13_true_error = all_errors.g13_true_error;
5.	fprintf(fp_g13_true_error, "%f\n", g13_true_error);
6.	fclose(fp_g13_true_error);

3.3.4. QDA

$$W_{QDF}(x)-c=-\frac{1}{2}(x-\bar{x}_0)^T\underbrace{(\alpha\times I_p+C_0)^{-1}}_{K_0}(x-\bar{x}_0)+\frac{1}{2}(x-\bar{x}_1)^T\underbrace{(\alpha\times I_p+C_1)^{-1}}_{K_1}(x-\bar{x}_1)$$

$$+\frac{1}{2}\log\left(\frac{|\alpha\times I_p+C_1|}{|\alpha\times I_p+C_0|}\right)-c$$

$$=-\frac{1}{2}(x^TK_0x-x^TK_0\bar{x}_0-\bar{x}_0^TK_0x+\bar{x}_0^TK_0\bar{x}_0)$$

$$+\frac{1}{2}(x^TK_1x-x^TK_1\bar{x}_1-\bar{x}_1^TK_1x+\bar{x}_1^TK_1\bar{x}_1)+\frac{1}{2}\log\left(\frac{|\alpha\times I_p+C_1|}{|\alpha\times I_p+C_0|}\right)-c$$

In our case $(\boldsymbol{vector1})^T\times\boldsymbol{matrix}_{\boldsymbol{square}}\times(\boldsymbol{vector2})$ is scalar(just a number, not matrix), then it follows that

$$(\boldsymbol{vector1})^T\times\boldsymbol{matrix}_{\boldsymbol{square}}\times(\boldsymbol{vector2})=\left((vector1)^T\times matrix_{square}\times(vector2)\right)^T$$

$$=(vector2)^T\times matrix_{square}^T\times(vector1)$$

So,

$$\boldsymbol{\bar{x}_0^TK_0x}=(\bar{x}_0^TK_0x)^T=\boldsymbol{x^TK_0^T\bar{x}_0}$$

$$\boldsymbol{\bar{x}_1^TK_1x}=(\bar{x}_1^TK_1x)^T=\boldsymbol{x^TK_1^T\bar{x}_1}$$

Matrix M is said symmetric, if $M=M^T$. Covariance matrix is positive semi-definite which means symmetric too. And identity matrix is symmetric matrix. And inverse of sum of symmetric matrices is symmetric matrix too Kernel Methods n.d.). So, $K_0=(\alpha\times I_p+C_0)^{-1}$, $K_1=(\alpha\times I_p+C_1)^{-1}$ are symmetric. So, $K_0^T=K_0$, $K_1^T=K_1$.

Then it follows:

$$\boxed{\bar{x}_0^TK_0x=x^TK_0\bar{x}_0}$$

$$\boxed{\bar{x}_1^TK_1x=x^TK_1\bar{x}_1}$$

So, after substituting it follows:

$$W_{QDF}(x)-c=\underbrace{x^T\underbrace{\left(\frac{K_1-K_0}{2}\right)}_{qda.a}x+x^T\underbrace{(K_0\bar{x}_0-K_1\bar{x}_1)}_{qda.b}+0.5\underbrace{\left(\bar{x}_1^TK_1\bar{x}_1-\bar{x}_0^TK_0\bar{x}_0+\log\left(\frac{|\alpha\times I_p+C_1|}{|\alpha\times I_p+C_0|}\right)\right)-c}_{qda.c}}_{the\ code\ was\ written\ according\ to\ this}$$

classifier.h (qda):

1. It declares the struct QDA with `double **a,` which holds a matrix, and with `double *b,` which holds a vector, and with `double c,` which holds constant number.

2. Prototype of functions: void qdaTrn, double qdaTst.

#	Codes

1.	`struct QDA {`
	`    double **a;`
	`    double *b;`
	`    double c;`
	`};`

2.	`void qdaTrn(double** X, int* y, int N, int d, int* ind, QDA* qda);`
	`double qdaTst(double** X, int* y, int N, int d, int* ind, QDA qda);`

classifier.cpp (qda)

To make qda.Trn and qda.Tst, code of lda.Trn and lda.Tst were copied and pasted after g13.Tst. Then all keywords, that indicate lda and LDA were changed to qda and QDA. qda.a, qda.b, qda.c were calculated using functions from matrixOperations.cpp following the equation of QDA above (you can see how they were calculated, downloading classifier.cpp). When calculating qda.c, to calculate the determinant, `determinant` function was not used due to it takes too much time to calculate the determinant when d>10. Because, `determinant` function is recursive function, so it too much increases the number of operations. So, to calculate determinant, `chol_dc` function and **`for`** loop were used.

1. `chol_dc` function makes Cholesky factor of the given positive-definite matrix. In our case the covariance matrix is positive semi-definite and identity matrix is positive-definite. So, $\alpha \times I_p$ is positive-definite, because α=0.1 is positive (Hyper-Textbook: Optimization Models and application 2014). Consequntly, $\alpha \times I_p + C_0$ and $\alpha \times I_p + C_1$ are positive-definite. So, `chol_dc` function can be used.
2. $M = L \times L^T$, where M is positive-definite matrix and L is Cholesky factor which is lower triangular matrix with positive elements. Mathematically, $|M| = |L \times L^T| = |L| \times |L^T|$. Determinant of lower and upper triangular matrix is the product of its diagonal elements. And diagonal elements of L and L^T are equal. So, $|M| = |L| \times |L^T| = |L|^2$ is the product of square of each diagonal element of L (Cholesky 2011).
And **`for`** loop mulptiplies square of each diagonal element of `for_determinant_0` and `for_determinant_1`, and initializes them to `determinant_0` and `determinant_1` respectively.

#	Codes

1.	`chol_dc(cov_hat_0, d, for_determinant_0);`
	`chol_dc(cov_hat_1, d, for_determinant_1);`

2.	for(i = 0; i<d; i++) {
	    determinant_0 = determinant_0 * for_determinant_0[i][i] * for_determinant_0[i][i];
	    determinant_1 = determinant_1 * for_determinant_1[i][i] * for_determinant_1[i][i];
	}

calculateErrors.h (qda)

1. qda_true_error is declared as double and it holds the value of the true error of QDA.

#	Codes
1.	double qda_true_error;

Download calculateErrors.cpp (qda)

1. It declares model_QDA as variable of struct QDA.

2. The (*all_errors).qda_true_error is initialized to 0.

3. The 2D model_QDA.a was made, the size of model_QDA.b array is initialized to d, and it can be deleted from the memory because of it is new double.

4. qdaTrn function is invoked, and main purpose of qdaTrn is to calculate the value of model_QDA.a, model_QDA.b, model_QDA.c.

5. (*all_errors).qda_true_error takes the return value of qdaTst function.

6. model_QDA.a array is deleted from the memory.

#	Codes
1.	QDA model_QDA;
2.	(*all_errors).qda_true_error = 0.00;

3.	model_QDA.a = make_2D_matrix(d, d, dinit);
	model_QDA.b = new double [d];

4.	qdaTrn(data_trn.data, data_trn.labels, data_trn.N, d, best_features, &model_QDA);
5.	(*all_errors).qda_true_error = qdaTst(data_tst.data, data_tst.labels, data_tst.N, d, best_features, model_QDA);
6.	delete model_QDA.a;

main.cpp (qda)

1. The value of double qda_true_error after initiliztion will be written in output .txt file.

2. Declaration of FILE* fp_qda_true_error;

3. It makes the file with the name `*_qda_true_error_round=1.txt,` where `*` is given by the command line.

4. `qda_true_error` is initialized.

5. The value of `qda_true_error` is written in `*_qda_true_error_round=1.txt` file.

6. File is closed.

#	Codes
1.	`double qda_true_error;`
2.	`FILE* fp_qda_true_error;`

3.	`sprintf(outputFilenameFinal, "%s_qda_true_error_round=%d.txt",`
	`outputFilename, round);`
	`fp_qda_true_error = fopen(outputFilenameFinal, "w");`
	`if(fp_qda_true_error==NULL)`
	`{`
	`    printf("cannot open %s_qda_true_error_round=%d.txt",`
	`outputFilename, round);`
	`    return(-1);`
	`}`

4.	`qda_true_error = all_errors.qda_true_error;`
5.	`fprintf(fp_qda_true_error, "%f\n", qda_true_error);`
6.	`fclose(fp_qda_true_error);`

3.3.5. SEDC

Expanding parenthesis for coding

$$W_{SEDC}(x) - c = m\left(z - \frac{\bar{z}_0 + \bar{z}_1}{2}\right)^T (\bar{z}_0 - z_1) - c = \underbrace{z^T \underbrace{(\bar{z}_0 - \bar{z}_1)m}_{sedc.a} \underbrace{-0.5(\bar{z}_0 + \bar{z}_1)^T(\bar{z}_0 - \bar{z}_1)m - c}_{sedc.b}}_{the\ code\ was\ written\ according\ to\ this}$$

Source files:

Download classifier.h (sedc):

1. It declares the struct SEDC with `double a` and which holds scalar constant value of $W_{SEDC}(x) - c$ indicated as *sedc.b.*

2. Prototype of functions: void sedcTrn, double sedcTst.

#	Codes

1.	`struct SEDC {`
	`    double a;`
	`    double b;`
	`};`

2.	`void sedcTrn(double** X, int* y, int N, int d, int* ind, SEDC* sedc);`
	`double sedcTst(double** X, int* y, int N, int d, int* ind, SEDC sedc);`

classifier.cpp (sedc)

To make sedc.Trn and sedcTst, code of ldaTrn and ldaTst were copied and pasted after ldaTst. Then all keywords, that indicate lda and LDA were changed to sedc and SEDC. Then arrays of pooled covariance, inverted matrix were removed.

1. `(*sedc).a` and `(*sedc).b` are calculated.

#	Codes

1.	`for(i = 0; i<d; i++) {`
	`     z_0 += mu_hat_0[i]; //z_0 is just sum of elements of mu_hat_0`
	`     z_1 += mu_hat_1[i]; //z_1 is just sum of elements of mu_hat_1`
	`}`
	
	`(*sedc).a = z_0 - z_1; //(*sedc).a is ready`
	`(*sedc).b = (z_0 - z_1) * (-(z_0/d + z_1/d))/2 - log((double)N_1/(double)N_0);`

calculateErrors.h (sedc)

1. `sedc_true_error` is declared as double and it holds the value of the true error of SEDC.

#	Codes
1.	`double sedc_true_error;`

calculateErrors.cpp (sedc)

1. It declares `model_SEDC` as variable of struct `SEDC`.

2. The `(*all_errors).sedc_true_error` is initialized to `0`.

3. `sedcTrn` function is invoked, and main purpose of `sedcTrn` is to calculate the value of `model_SEDC.a` and `model_SEDC.b`

4. `(*all_errors).sedc_true_error` takes the return value of `sedcTst` function.

#	Codes
1.	`SEDC model_SEDC;`
2.	`(*all_errors).sedc_true_error = 0.00;`
3.	`sedcTrn(data_trn.data, data_trn.labels, data_trn.N, d, best_features, &model_SEDC);`
4.	`(*all_errors).sedc_true_error = sedcTst(data_tst.data, data_tst.labels, data_tst.N, d, best_features, model_SEDC);`

main.cpp (sedc)

1. The value of `double sedc_true_error` after initiliztion will be written in output .txt file.

2. Declaration of `FILE* fp_sedc_true_error;`

3. It makes the file with the name *_sedc_true_error_round=1.txt, where * is given by the command line.

4. sedc_true_error is initialized.

5. The value of sedc_true_error is written in *_sedc_true_error_round=1.txt file.

6. File is closed.

#	Codes
1.	double sedc_true_error;
2.	FILE* fp_sedc_true_error;

3.	sprintf(outputFilenameFinal, "%s_sedc_true_error_round=%d.txt", outputFilename, round);
	fp_sedc_true_error = fopen(outputFilenameFinal, "w");
	if(fp_sedc_true_error==NULL)
	{
	    printf("cannot open %s_sedc_true_error_round=%d.txt", outputFilename, round);
	    return(-1);
	}

4.	sedc_true_error = all_errors.sedc_true_error;
5.	fprintf(fp_sedc_true_error, "%f\n", sedc_true_error);
6.	fclose(fp_sedc_true_error);

3.3.6. RLDA

Expanding parenthesis for coding

$$c - W_{RLDA}(x) = c - \left(x - \frac{\bar{x}_0 + \bar{x}_1}{2}\right)^T (\gamma \times I_p + C)^{-1}(\bar{x}_0 - \bar{x}_1)$$

$$= \underbrace{x^T \underbrace{(\gamma \times I_p + C)^{-1}(\bar{x}_1 - \bar{x}_0)}_{rlda.a} \underbrace{-0.5(\bar{x}_0 + \bar{x}_1)^T \underbrace{(\gamma \times I_p + C)^{-1}(\bar{x}_1 - \bar{x}_0)}_{rlda.a} + c}_{rlda.b}}_{the\ code\ was\ written\ according\ to\ this}$$

<u>Source files:</u>

classifier.h (rlda):

1. It declares the struct RLDA with **double** *a, which holds vector, and with **double** b, which holds scalar constant value of $c - W_{RLDA}(x)$ indicated as *rlda.b*.

2. Prototype of functions: void rldaTrn, double rldaTst.

#	Codes

1.	struct RLDA {
	    double *a;
	    double b;
	};

2.	void rldaTrn(double** X, int* y, int N, int d, int* ind, RLDA* rlda, int i);
	double rldaTst(double** X, int* y, int N, int d, int* ind, RLDA rlda);

classifier.cpp (rlda)

To make rlda.Trn and rldaTst, code of ldaTrn and ldaTst were copied and pasted after ldaTst. Then all keywords, that indicate lda and LDA were changed to rlda and RLDA.

1. int i was added to change the γ every time. γ_opt was found. Pooled cov becomes regularized.

```
#   Codes
1.  void rldaTrn(double** X, int* y, int N, int d, int* ind, RLDA* rlda,
    int i){
    .
    .
    double y_opt;
    y_opt = 0.001 * pow(2.00, i);
    .
    .
    for(k=0; k<d; k++) {
            pooled_cov[k][k] = pooled_cov[k][k] + y_opt;
    }
    .
    .
    }
```

calculateErrors.h (rlda)

1. `rlda_true_error` is declared as double and it holds the value of the true error of RLDA.

```
#   Codes
1.  double rlda_true_error;
```

calculateErrors.cpp (rlda)

```
#   Codes
1.  RLDA model_RLDA;

        (*all_errors).rlda_true_error = 0.00;

        SimulationData S_1; //S_1 is training data, which is indicated with [2N/3]
        SimulationData S_2; //S_2 is training data, which is indicated with N-[2N/3]

        int* best_features_opt; //best_features_opt will be found using the data_trn.data

        S_1.data = make_2D_matrix((2*N_trn/3), D, dinit);
        S_1.labels = new int [2*N_trn/3];
        S_2.data = make_2D_matrix((N_trn-2*N_trn/3), D, dinit);
        S_2.labels = new int [N_trn-2*N_trn/3];

        dataGeneration(data_trn.data, data_trn.labels, (2*N_trn/3), (N_trn-2*N_trn/3), D,
    seed, &S_1, &S_2); //training data is divided into S_1 and S_2

        best_features_opt = new int [d];
        featureSelection(S_1.data, S_1.labels, S_1.N, S_1.D, d, best_features_opt);
```

	int i;
	int y_opt_index;
	double e[21];
	double minimum_error;
	
	for(i = 0; i<21; i++) {
	model_RLDA.a = new double [d];
	
	rldaTrn(S_1.data, S_1.labels, S_1.N, d, best_features_opt, &model_RLDA, i);
	e[i] = rldaTst(S_2.data, S_2.labels, S_2.N, d, best_features_opt, model_RLDA);
	
	if(i==0)
	minimum_error = e[i];
	if(e[i]<minimum_error) {
	minimum_error = e[i];
	y_opt_index = i; //y_opt_index is the index of y_opt inside the rldaTrn
	}
	
	delete model_RLDA.a;
	}
	
	model_RLDA.a = new double [d];
	rldaTrn(data_trn.data, data_trn.labels, data_trn.N, d, best_features, &model_RLDA, y_opt_index); //y_opt_index is used to find the true error
	
	(*all_errors).rlda_true_error = rldaTst(data_tst.data, data_tst.labels, data_tst.N, d, best_features, model_RLDA);

2.	delete model_RLDA.a;

main.cpp (rlda)

1. The value of double rlda_true_error after initiliztion will be written in output .txt file.

2. Declaration of FILE* fp_rlda_true_error;

3. It makes the file with the name *_rlda_true_error_round=1.txt, where * is given by the command line.

4. rlda_true_error is initialized.

5. The value of rlda_true_error is written in *_rlda_true_error_round=1.txt file.

6. File is closed.

#	Codes
1.	double rlda_true_error;
2.	FILE* fp_rlda_true_error;

3.	sprintf(outputFilenameFinal, "%s_rlda_true_error_round=%d.txt", outputFilename, round);
	fp_rlda_true_error = fopen(outputFilenameFinal, "w");
	if(fp_rlda_true_error==NULL)
	{
	printf("cannot open %s_rlda_true_error_round=%d.txt", outputFilename, round);
	return(-1);
	}

4.	rlda_true_error = all_errors.rlda_true_error;

```
fprintf(fp_rlda_true_error, "%f\n", rlda_true_error);
fclose(fp_rlda_true_error);
```

REFERENCES:

A.Webb, R. Duda and S. Gong. n.d “Linear Discriminant Analysis(LDA).” Accessed November 7, 2016.
https://www.cse.unr.edu/~bebis/MathMethods/LDA/lecture.pdf

Cholesky factorization. 2011. Accessed November 7, 2016.
http://www.seas.ucla.edu/~vandenbe/103/lectures/chol.pdf

Hyper-Textbook: Optimization Models and Applications. 2014. “Positive Semi-Definite Matrices.” Accessed November 7, 2016.
https://inst.eecs.berkeley.edu/~ee127a/book/login/l_sym_psd.html

Jason, Weston. n.d. *Support Vector Machine.* Accessed November 7, 2016.
http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf

Keith, Baggerly, Kevin Coombes. 2004. “Analysis of Microarray Data.” Accessed November 7, 2016.
http://bioinformatics.mdanderson.org/MicroarrayCourse/Lectures/ma20b.pdf

OpenCV. n.d. “Introduction to Support Vector Machines” Last modified November 21, 2016.
http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

Pang, Herbert, Tong, Tiejun and Ng Michael. 2014. “Block-diagonal discriminant analysis and its bias-corrected rules.” De Gruyter(3).DOI: 10.1515/sagmb-2012-0017 Accessed November 7, 2016.
http://pubmedcentralcanada.ca/pmcc/articles/PMC3856189/

Ricardo, Gutierrez-Osuna. n.d. “Dimensionalty reduction (LDA).” Accessed November 7, 2016. http://courses.cs.tamu.edu/rgutier/cs790_w02/l6.pdf

ScikitLearn. n.d. “Support Vector Machines.” Accessed November 7, 2016.
http://scikit-learn.org/stable/modules/svm.html

Serdobolskii V.I. 2010. Discriminant analysis of high-dimensional data over limited samples. New York: Springer Publishing Company. doi: 10.1134/s1064562410010217.
http://link.springer.com/article/10.1134%2FS1064562410010217

Teknomo,Kardi.2015. ”Linear Discriminant Analysis.” Accessed November 7, 2016. http://people.revoledu.com/kardi/tutorial/LDA/LDA.html

Trevor, Hastie, Robert, Tibshirani and Jerome, Friedman. 2009. The elements of statistical learning. New York: Springer Publishing Company. PDF e-book. http://statweb.stanford.edu/~tibs/ElemStatLearn/printings/ESLII_print10.pdf

Village, Idiot, R. Berwick. n.d. *An Idiot's guide to Support vector machines (SVMs).* Accessed November 7, 2016. http://www.svms.org/tutorials/Berwick2003.pdf

Zollanvari, Amin. 2010. "Analytic Study of perfprmance of error estimators for linear discriminant analysis with applications in genomics." PhD dissertation, Texas A&M University. Accessed November 7, 2016. http://oaktrust.library.tamu.edu/bitstream/handle/1969.1/ETD-TAMU-2010-12-8685/ZOLLANVARI-DISSERTATION.pdf

Kernel Methods. n.d. "Symmetric, Positive Definite Matrices." Accessed November 7, 2016. http://math.iit.edu/~openscholar/meshfree/pages/symmetric-positive-definite-matrices