CS 679: Pattern Recognition

University of Nevada, Reno - Spring 2024

Assignment 4

Jaleesa Houle

Due: May 7th, 2024

Statement: "I declare that all material in this assignment is my own work except where there is clear acknowledgment or reference to the work of others. I understand that both my report and code may be subjected to plagiarism detection software, and fully accept all consequences if found responsible for plagiarism, as explained in the syllabus, and described in UNR's Academic Standards Policy: UAM 6,502."

1. Theory

Experiment 1

General Backround

For this assignment, we are asked to explore gender classification of given images using Support Vector Machines (SVM). PCA was performed on the images, and the corresponding eigenvalues, eigenvectors, and eigen-coefficients were found according to the theory discussed in assignment 3. To perform classification, the top 30 eigen-coefficients associated with each image will be used. When implementing SVM, it is best practice to scale the data so that it lies within a standard range of values. For this assignment, SVM will be implemented using the Lib-SVM python library, and the data will be scaled so that all points are between [-1,1] prior training and testing each model.

To test various models of SVM, we will be implementing a three-fold cross validation by training models on three different sets of training data and then testing those models on corresponding validation data sets. Once the optimum set of parameters $(\gamma_{\text{opt}}, C_{\text{opt}})$ are found, they will be used to train models with three training data sets and then those models will be tested on three unseen testing data sets.

Support Vector Machines

SVM is a method of classification which seeks to minimize structural risk for given data. The idea is that a decision boundary or hyperplane will generalize better when there is a wider margin between the boundary and the nearest data points. The data points which are closest to the boundary are called Support Vectors (SVs), as they are the most influential when determining the decision boundary. Training an SVM involves solving a quadratic problem with linear constraints.

There are three general cases in which SVMs can be used. The most simple case is when the data is linearly separable. In this instance, we can consider the general form of the linear discriminant,

$$g(x) = w^T x + w_0 \tag{1}$$

where we decide class 1 if $g(x) > 0$ and class 2 if $g(x) < 0$. If we consider the dual classification problem

$$z_k(w^T x_k + w_0) > 0 \tag{2}$$

for $k = 1, 2, ..., n$ data points, then we can see that the distance r between x_k and the decision boundary can be constrained such that

$$r = \frac{z_k g(x_k)}{||w||} > b \tag{3}$$

for $b > 0$ and $b||w|| > 1$. This then indicates that $b = 1/||w||$, where $2b$ is the total width of the margin separating the closest x_k data points on either side of the

decision boundary. The goal of SVM is then to maximize $2/||w||$, which can also be accomplished by minimizing the quadratic function

$$\frac{1}{2}||w||^2 \tag{4}$$

which is subject to $z_k g(x_k) > 1$ for $k = 1, 2, ...n$. In practice, this is done using Langrange optimization such that

$$L(w, w_0, \lambda) = \frac{1}{2}||w||^2 - \sum_{k+1}^{n} \lambda_k[(w^T x_k + w_0) - 1] \tag{5}$$

where $\lambda_k \geq 0$ are referred to as Langrange multipliers. Using these Langrange multipliers, we are able to reframe the problem so that we are maximizing

$$\sum_{k+1}^{n} \lambda_k - \frac{1}{2} \sum_{k,j}^{n} \lambda_k \lambda_j z_k z_j x_j^T x_k. \tag{6}$$

When maximizing this expression, we assume that $\sum_{k=1}^{n} z_k \lambda_k = 0$ and $\lambda_k > 0$. Solving this maximization problem leads to a linear discriminant in the new form

$$\begin{aligned} g(x) &= \sum_{k+1}^{n} z_k \lambda_k (x_k^T x) + w_0 \\ &= \sum_{k+1}^{n} z_k \lambda_k (x \cdot x_k) + w_0. \end{aligned} \tag{7}$$

This linear discriminant now only depends on the support vectors, as the value of λ_k is zero for any x_k which is not a support vector.

Non-Linear SVM and Kernels

The theory for SVM can be expanded for data that is not linearly separable by mapping the data points x_k using some function $\Phi(x_k)$. The non-linear SVM then is mapped to dimension h and can be expressed as

$$g(x) = \sum_{k+1}^{n} z_k \lambda_x (\Phi(x) \cdot \Phi(x_k)) + w_0. \tag{8}$$

In practice, mapping to h dimensions is computationally expensive, and finding the function $\Phi(x)$ can be challenging. As a solution to this problem, kernels can be employed to simplify the computations. A kernel is a positive-definite, symmetric matrix defined as

$$K(x, x_k) = \Phi(x) \cdot \Phi(x_k). \tag{9}$$

Replacing this into Equation 8, the discriminant becomes

$$g(x) = \sum_{k+1}^{n} z_k \lambda_x K(x, x_k) + w_0. \tag{10}$$

This manipulation allows us to compute these dot products without explicitly mapping to the higher feature space. The challenge then becomes finding the kernel and its corresponding parameters which yield the best results for a given problem.

For this experiment, we will be exploring different parameters for the polynomial and the radial basis function (RBF) kernel. The polynomial kernel is defined as

$$K(x, x_k) = (\gamma x^T x_k + c_0)^d. \tag{11}$$

We are asked to explore the results of a three-fold cross validation experiment by varying the degree so that $d = [1, 2, 3]$. To simplify parameter options, we will set $\gamma = 1$ and $c_0 = 0$.

The RBF kernel is defined as

$$K(x, x_k) = e^{-\gamma |x - x_k|^2}. \tag{12}$$

For this kernel, we are asked to explore the results for $\gamma = [0.1, 1, 10, 100]$. In addition to varying γ and d for these two kernels, we are told to explore the result of choosing different cost values C, which are associated with an added cost function. For data which may contain outliers (i.e., the data is "almost" linearly separable), a cost function can be added to Equation 7 so that we can account for these misclassifications by introducing positive error variables Ψ_k so that

$$z_k(w^T x_k + w_0) \geq 1 - \Psi_k \tag{13}$$

for $k = 1, 2, ...n$. These variables are referred to as slack variables, and they provide a framework for a modified linear discriminant function by seeking to minimize

$$\frac{1}{2}||w||^2 + C \sum_{k+1}^{n} \Psi_k \tag{14}$$

where C is a constant associated with the cost of allowing misclassifications. By allowing these slack variables, the Langrange multipliers λ_k from equation 7 become constrained such that $0 < \lambda_k < C$. By observation of Equation 14, we can see that smaller values of C will result in a smaller overall value of $C \sum_{k+1}^{n} \Psi_k$, so the inclusion of misclassifications is treated less severely as we try to minimize Equation 14. Conversely, large values of C will make that expression much larger, associating a higher cost with misclassifications, which will lead to a decision boundary with a smaller margin of separation as we seek to minimize those misclassifications. One challenge of using SVM is determining the best cost so that these misclassifications

are minimized while the margin of separation is maximized. For this experiment, we will be exploring the effect of setting $C = [0.1, 1, 10, 100]$ for both the polynomial and RBF kernels.

Experiment 2

For experiment 2, we are asked to use Bayes Theory to classify the data used in experiment 1. We are told to use maximum likelihood estimation (MLE) to determine the parameters from the training data. Since Bayes theory and MLE are discussed in Assignments 1 and 2, the theory behind these approaches will be omitted here. In order to classify the data, we are told to assume the features are uncorrelated and to use the Mahalanobis distance for classification, which is defined as

$$g_i(x) = -\frac{1}{2}(\mathbf{x} - \mu_\mathbf{i})^T \Sigma^{-1} (\mathbf{x} - \mu_\mathbf{i}) \tag{15}$$

for each class i. Alternatively, we can write this as

$$\mathbf{w^T}(\mathbf{x} - \mathbf{x_0}) = 0 \tag{16}$$

where $w = \Sigma^{-1}(\mu_i - \mu_j)$ and $x_0 = 1/2(\mu_i + \mu_j)$ for $i = 1$ and $j = 2$ classes. To use the Mahalanobis distance, we assume that the covariance matrix for each class is the same. Upon looking at the ML estimated covariance matrices for class 1 and class 2, it was clear that the two covariance matrices were not equivalent. As such, I chose to use the covariance matrix associated with class 1 for this assignment, though it can be noted that using the covariance matrix for class 2 may yield different classification results. For this experiment, it was not necessary to scale the data between [-1,1] as in experiment 1.

2. Results and Discussion

Experiment 1

The results of varying the parameters d and C using the polynomial kernel are displayed in Table 1. We can see that the average results across Folds 1, 2, and 3 were best when using $d = 1$ and $d = 3$ for both the high resolution and low resolution images. Of the parameter choices examined, the lowest average misclassification rates were seen when $d = 1$ and $C = 0.1$ for both sets of images.

The results of varying parameters γ and C for the RBF kernel are shown in Table 2. Though there were parameters which performed similarly well when using both the RBF and polynomial kernels, the lowest average misclassification error was achieved when using the RBF kernel (7.27% and 9.02% for the low resolution and high resolution images, respectively).

Misclassification Results - Polynomial Kernel						
Training Data	**Degree**	**C**	**Fold 1**	**Fold 2**	**Fold 3**	**Average**
Low Resolution Images (16x20)	1	0.1	11.28	6.77	8.27	8.77
		1	9.02	11.28	8.27	9.52
		10	13.53	15.04	11.28	13.28
		100	9.77	14.29	11.28	11.78
	2	0.1	23.31	20.30	18.80	20.80
		1	22.56	20.30	23.31	22.06
		10	22.56	20.30	23.31	22.06
		100	22.56	20.30	23.31	22.06
	3	0.1	14.29	7.52	13.53	11.78
		1	14.29	7.52	13.53	11.78
		10	14.29	7.52	13.53	11.78
		100	14.29	7.52	13.53	11.78
High Resolution Images (48x60)	1	0.1	10.53	4.51	12.78	9.27
		1	9.02	8.27	23.31	13.53
		10	10.53	9.02	26.32	15.29
		100	10.53	17.29	26.32	18.05
	2	0.1	18.05	20.30	20.30	19.55
		1	16.54	23.31	21.05	20.30
		10	16.54	23.31	21.05	20.30
		100	16.54	23.31	21.05	20.30
	3	0.1	9.77	6.02	13.53	9.77
		1	9.77	6.02	13.53	9.77
		10	9.77	6.02	13.53	9.77
		100	9.77	6.02	13.53	9.77

Table 1: Classification results for $d = 1, 2, 3$ and $C = 0.1, 1, 10, 100$ using a polynomial kernel with $\gamma = 1$ and c0 $= 0$.

We can see that for both the low resolution and high resolution images, the best value of γ was 0.1. For the low resolution images, the average misclassification rates across Folds 1, 2, and 3 were the same when $C = 1$ and $C = 10$. Since the best classification rates for the high resolution data were found when setting $C = 1$, the parameters, used in testing were kept the same for both the low resolution and high resolution data. Thus, the RBF kernel with parameters $\gamma = 0.1$ and $C = 1$ were found to be the optimal choices $(\gamma_{\text{opt}}, C_{\text{opt}})$, and were used to classify the test set of images.

Classification Results - RBF Kernel						
Training Data	γ	**C**	**Fold 1**	**Fold 2**	**Fold 3**	**Average**
Low Resolution Images (16x20)	0.1	0.1	45.11	43.61	49.62	46.12
		1	8.27	6.02	7.52	7.27
		10	7.52	6.77	7.52	7.27
		100	7.52	7.52	7.52	7.52
	1	0.1	45.11	43.61	49.62	46.12
		1	28.57	22.56	39.10	30.08
		10	26.32	21.8	36.09	28.07
		100	26.32	21.8	36.09	28.07
	10	0.1	45.11	43.61	49.62	46.12
		1	45.11	43.61	49.62	46.12
		10	45.11	43.61	49.62	46.12
		100	45.11	43.61	49.62	46.12
	100	0.1	45.11	43.61	49.62	46.12
		1	45.11	43.61	49.62	46.12
		10	45.11	43.61	49.62	46.12
		100	45.11	43.61	49.62	46.12
High Resolution Images (48x60)	0.1	0.1	45.11	43.61	49.62	46.12
		1	9.02	5.26	12.78	9.02
		10	9.02	7.52	17.29	11.28
		100	9.02	8.27	17.29	11.53
	1	0.1	45.11	43.61	49.62	46.12
		1	28.57	22.56	41.35	30.83
		10	27.82	18.80	37.59	28.07
		100	27.82	18.80	37.59	28.07
	10	0.1	45.11	43.61	49.62	46.12
		1	45.11	43.61	49.62	46.12
		10	45.11	43.61	49.62	46.12
		100	45.11	43.61	49.62	46.12
	100	0.1	45.11	43.61	49.62	46.12
		1	45.11	43.61	49.62	46.12
		10	45.11	43.61	49.62	46.12
		100	45.11	43.61	49.62	46.12

Table 2: Misclassification rates using an RBF kernel with $\gamma = 0.1, 1, 10, 100$ and $C = 0.1, 1, 10, 100$ The lowest average error for high and low resolution images is highlighted in yellow.

For many parameter combinations, the error rates across folds was the same. This can be seen for values of $d = 2$ and $d = 3$ in Table 1 in the low and high resolution images. Additionally, we can see that the results for $\gamma = 10$ and $\gamma = 100$ are the same for all folds in Table 2. These results indicate that performance is unlikely to change significantly when the kernel and parameters are far off from the best fit

choices.

Image Resolution	Fold 1	Fold 2	Fold 3	Average
16x20	9.02	9.02	9.77	9.27
48x60	19.55	8.27	13.53	13.78

Table 3: Misclassification rates for each test set of images, using the same training data for each fold and the best parameters. For both the low and high resolution images, the best parameters were found using the RBF kernel with $\gamma = 0.1$ and $C = 1$.

.

Table 3 demonstrates the misclassification rates for Folds 1, 2, and 3 when implementing SVM using the RBF kernel with ($\gamma_{\text{opt}} = 0.1, C_{\text{opt}} = 1$) We can see that the average error rate for the low resolution images is slightly higher (9.27%) that the average found during parameter exploration (7.27% in Table 2). The same is true for the high resolution images, which had an average classification error rate of 9.02% during the parameter exploration phase versus an overall average error of 13.78% during testing. Overall, the classification success for both the low and high resolution images was relatively high, indicating that the model is able to generalize reasonably well when testing unseen data.

Experiment 2

Image Resolution	Fold 1	Fold2	Fold3	Average
16x20	11.28	6.02	10.53	9.27
48x60	9.02	6.77	11.28	9.02

Table 4: Misclassification rates for each test set of images using Bayesian classification. For these experiments, the distributions for each class were assumed to be multivariate Gaussian, and the covariance matrices were assumed to uncorrelated and the same for each class.

The resulting misclassification rates using Bayes Theory are shown in Table 4. We can see that, on average, the Bayes classifier performed the same for the low resolution images as the SVM classifier (Table 3). Though the average was the same, there was more variation across folds, indicating that these results may be slightly less consistent than the SVM classification. For the high resolution data, the Bayes classifier outperformed the SVM model (9.02% versus 13.78%, respectively). The better performance using Bayes classification indicates that our assumptions that the data is Gaussian are reasonable. Additionally, since we used the Mahalanobis distance for classification, it can be inferred that the features representing the male and female classes are able to be linearly separated reasonably well.

References

Duda, R. O., Hart, P. E., et al. (2006). *Pattern classification*. John Wiley & Sons.

Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification.

Appendix

Code

Instructions to run code and generate figures:

All code used to generate the data and figures in this assignment is included in separate files which consist of the following:

For Experiment 1:

'SVM_functions.py': This script contains helper functions for performing SVM using Lib-SVM with various parameters.

'CS679_Assignment4_JaleesaHoule_Experiment1.ipynb': The actual data generation and analysis for Experiment 1 was performed in a Jupyter Notebook environment. A pdf of this notebook is also included in the pages below.

For Experiment 2:

'MaxLikelihoodEstimator.py': This is a modified script from Assignment 2 which performs MLE on given data, assuming a Gaussian distribution.

'BayesianClassifier.py': This is a modified script from Assignment 1 which performs Bayesian classification on given data, assuming a Gaussian distribution.

'CS679_Assignment4_JaleesaHoule_Experiment2.ipynb': The actual data generation and analysis for Experiment 2 was performed in a Jupyter Notebook environment. A pdf of this notebook is also included in the pages below.

To run this code, download the python scripts and the Jupyter Notebook files into the same directory. The given directory 'GenderDataRowOrder' should be downloaded to the same directory as well. Open the Jupyter Notebooks for Experiments 1 and 2 and select 'run all'. This should re-run all of the analyses conducted for this assignment. The code requires installation of packages SciPy, Numpy, Pandas, Sympy, Scikit-learn, and Lib-SVM. This code was run using Python 3.8.15.

CS679_Assignment4_JaleesaHoule_Experiment1

May 3, 2024

[1]:
```
import numpy as np
import scipy
import pandas as pd
from libsvm.svmutil import *
from sklearn.preprocessing import MinMaxScaler

import SVM_functions
```

1 Experiment 1: Apply Support Vector Machines (SVMs) for gender classification. Experiment both with polynomial and RBF kernels as well as with different C values. Show your results both for 16x20 and 48x60 size images.

1.0.1 Naming convention:

EigenVectors_xx: eigenvectors of training images for fold xx

EigenValues_xx: eigenvalues of training images for fold xx

trPCA_xx: PCA projected training images for fold xx

TtrPCA_xx: labels of projected training images for fold xx

valPCA_xx: PCA projected validation images for fold xx

TvalPCA_xx: labels of projected validation images for fold xx

tsPCA_xx: PCA projected test images for fold xx

TtsPCA_xx: labels of projected test images for fold xx

2 Fold 1

2.1 High Resolution

[2]:
```
# training data
train_eigencoefficients_f1  = np.loadtxt('GenderDataRowOrder/48_60/trPCA_01.
 ↪txt')
train_class_labels_f1  = np.loadtxt('GenderDataRowOrder/48_60/TtrPCA_01.txt')

# validation data

val_eigencoefficients_f1 = np.loadtxt('GenderDataRowOrder/48_60/valPCA_01.txt')
val_class_labels_f1  = np.loadtxt('GenderDataRowOrder/48_60/TvalPCA_01.txt')
```

2.1.1 Polynomial kernel

[3]:
```
fold1_poly_results, fold1_poly_misclassifications = SVM_functions.
 ↪train_model(train_eigencoefficients_f1[:,:30], train_class_labels_f1,␣
 ↪val_eigencoefficients_f1[:,:30], val_class_labels_f1 , kernel='poly')
```

2.1.2 RBF kernel

[4]:
```
fold1_RBF_results, fold1_RBF_misclassifications = SVM_functions.
 ↪train_model(train_eigencoefficients_f1[:,:30], train_class_labels_f1,␣
 ↪val_eigencoefficients_f1[:,:30], val_class_labels_f1 , kernel='RBF')
```

2.2 Low Resolution

[5]:
```
# training data
train_eigencoefficients_f1_lowres  = np.loadtxt('GenderDataRowOrder/16_20/
 ↪trPCA_01.txt')
train_class_labels_f1_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtrPCA_01.
 ↪txt')

# validation data

val_eigencoefficients_f1_lowres = np.loadtxt('GenderDataRowOrder/16_20/
 ↪valPCA_01.txt')
val_class_labels_f1_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TvalPCA_01.
 ↪txt')
```

2.2.1 Polynomial kernel

[6]:
```
fold1_poly_results_lowres, fold1_poly_misclassifications_lowres = SVM_functions.
 ↪train_model(train_eigencoefficients_f1_lowres[:,:30],␣
 ↪train_class_labels_f1_lowres, val_eigencoefficients_f1_lowres[:,:30],␣
 ↪val_class_labels_f1_lowres , kernel='poly')
```

2.2.2 RBF kernel

[7]:
```
fold1_RBF_results_lowres, fold1_RBF_misclassifications_lowres = SVM_functions.
↪train_model(train_eigencoefficients_f1_lowres[:,:30],␣
↪train_class_labels_f1_lowres, val_eigencoefficients_f1_lowres[:,:30],␣
↪val_class_labels_f1_lowres , kernel='RBF')
```

3 Fold 2

3.1 High Resolution

[8]:
```
train_eigencoefficients_f2  = np.loadtxt('GenderDataRowOrder/48_60/trPCA_02.
↪txt')
train_class_labels_f2  = np.loadtxt('GenderDataRowOrder/48_60/TtrPCA_02.txt')

val_eigencoefficients_f2 = np.loadtxt('GenderDataRowOrder/48_60/valPCA_02.txt')
val_class_labels_f2  = np.loadtxt('GenderDataRowOrder/48_60/TvalPCA_02.txt')
```

3.1.1 Polynomial kernel

[9]:
```
fold2_poly_results, fold2_poly_misclassifications = SVM_functions.
↪train_model(train_eigencoefficients_f2[:,:30], train_class_labels_f2,␣
↪val_eigencoefficients_f2[:,:30], val_class_labels_f2 , kernel='poly')
```

3.1.2 RBF kernel

[10]:
```
fold2_RBF_results, fold2_RBF_misclassifications = SVM_functions.
↪train_model(train_eigencoefficients_f2[:,:30], train_class_labels_f2,␣
↪val_eigencoefficients_f2[:,:30], val_class_labels_f2 , kernel='RBF')
```

3.2 Low Resolution

[11]:
```
train_eigencoefficients_f2_lowres  = np.loadtxt('GenderDataRowOrder/16_20/
↪trPCA_02.txt')
train_class_labels_f2_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtrPCA_02.
↪txt')

val_eigencoefficients_f2_lowres = np.loadtxt('GenderDataRowOrder/16_20/
↪valPCA_02.txt')
val_class_labels_f2_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TvalPCA_02.
↪txt')
```

3.2.1 Polynomial kernel

[12]:
```
fold2_poly_results_lowres, fold2_poly_misclassifications_lowres = SVM_functions.
↪train_model(train_eigencoefficients_f2_lowres[:,:30],␣
↪train_class_labels_f2_lowres, val_eigencoefficients_f2_lowres[:,:30],␣
↪val_class_labels_f2_lowres , kernel='poly')
```

3.2.2 RBF kernel

[13]:
```
fold2_RBF_results_lowres, fold2_RBF_misclassifications_lowres = SVM_functions.
↪train_model(train_eigencoefficients_f2_lowres[:,:30],␣
↪train_class_labels_f2_lowres, val_eigencoefficients_f2_lowres[:,:30],␣
↪val_class_labels_f2_lowres , kernel='RBF')
```

4 Fold 3

4.1 High Resolution

[14]:
```
train_eigencoefficients_f3  = np.loadtxt('GenderDataRowOrder/48_60/trPCA_03.
↪txt')
train_class_labels_f3  = np.loadtxt('GenderDataRowOrder/48_60/TtrPCA_03.txt')

val_eigencoefficients_f3 = np.loadtxt('GenderDataRowOrder/48_60/valPCA_03.txt')
val_class_labels_f3  = np.loadtxt('GenderDataRowOrder/48_60/TvalPCA_03.txt')
```

4.1.1 Polynomial kernel

[15]:
```
fold3_poly_results, fold3_poly_misclassifications = SVM_functions.
↪train_model(train_eigencoefficients_f3[:,:30], train_class_labels_f3,␣
↪val_eigencoefficients_f3[:,:30], val_class_labels_f3 , kernel='poly')
```

4.1.2 RBF kernel

[16]:
```
fold3_RBF_results, fold3_RBF_misclassifications = SVM_functions.
↪train_model(train_eigencoefficients_f3[:,:30], train_class_labels_f3,␣
↪val_eigencoefficients_f3[:,:30], val_class_labels_f3 , kernel='RBF')
```

4.2 Low Resolution

[17]:
```
train_eigencoefficients_f3_lowres  = np.loadtxt('GenderDataRowOrder/16_20/
↪trPCA_03.txt')
train_class_labels_f3_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtrPCA_03.
↪txt')

val_eigencoefficients_f3_lowres = np.loadtxt('GenderDataRowOrder/16_20/
↪valPCA_03.txt')
```

```
val_class_labels_f3_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TvalPCA_03.
 ↪txt')
```

4.2.1 Polynomial kernel

[18]:
```
fold3_poly_results_lowres, fold3_poly_misclassifications_lowres = SVM_functions.
 ↪train_model(train_eigencoefficients_f3_lowres[:,:30],␣
 ↪train_class_labels_f3_lowres, val_eigencoefficients_f3_lowres[:,:30],␣
 ↪val_class_labels_f3_lowres , kernel='poly')
```

4.2.2 RBF kernel

[19]:
```
fold3_RBF_results_lowres, fold3_RBF_misclassifications_lowres = SVM_functions.
 ↪train_model(train_eigencoefficients_f3_lowres[:,:30],␣
 ↪train_class_labels_f3_lowres, val_eigencoefficients_f3_lowres[:,:30],␣
 ↪val_class_labels_f3_lowres , kernel='RBF')
```

5 Find the optimum set of parameters (opt, Copt)

5.0.1 High Resolution Images

[20]:
```
misclassifications_highres = [np.
 ↪concatenate([fold1_poly_misclassifications,fold1_RBF_misclassifications]),␣
 ↪np.
 ↪concatenate([fold2_poly_misclassifications,fold2_RBF_misclassifications]),␣
 ↪np.concatenate([fold3_poly_misclassifications,fold3_RBF_misclassifications])]
kernel_summaries_highres= [np.concatenate([fold1_poly_results,␣
 ↪fold1_RBF_results]), np.concatenate([fold2_poly_results,␣
 ↪fold2_RBF_results]), np.concatenate([fold3_poly_results, fold3_RBF_results])]

high_res_summarydf = SVM_functions.
 ↪get_optimum_params(misclassifications_highres, kernel_summaries_highres)
```

```
Best average error:  [9.02255639]
Best parameters:  ['RBF, gamma= 0.1, C= 1']
```

[21]:
```
high_res_summarydf
```

[21]:

	Params	Fold1	Fold2	Fold3	Average
0	Polynomial, d= 1, C= 0.1	10.526316	4.511278	12.781955	9.273183
1	Polynomial, d= 1, C= 1	9.022556	8.270677	23.308271	13.533835
2	Polynomial, d= 1, C= 10	10.526316	9.022556	26.315789	15.288221
3	Polynomial, d= 1, C= 100	10.526316	17.293233	26.315789	18.045113
4	Polynomial, d= 2, C= 0.1	18.045113	20.300752	20.300752	19.548872
5	Polynomial, d= 2, C= 1	16.541353	23.308271	21.052632	20.300752
6	Polynomial, d= 2, C= 10	16.541353	23.308271	21.052632	20.300752
7	Polynomial, d= 2, C= 100	16.541353	23.308271	21.052632	20.300752

8	Polynomial, d= 3, C= 0.1	9.774436	6.015038	13.533835	9.774436
9	Polynomial, d= 3, C= 1	9.774436	6.015038	13.533835	9.774436
10	Polynomial, d= 3, C= 10	9.774436	6.015038	13.533835	9.774436
11	Polynomial, d= 3, C= 100	9.774436	6.015038	13.533835	9.774436
12	RBF, gamma= 0.1, C= 0.1	45.112782	43.609023	49.624060	46.115288
13	RBF, gamma= 0.1, C= 1	9.022556	5.263158	12.781955	9.022556
14	RBF, gamma= 0.1, C= 10	9.022556	7.518797	17.293233	11.278195
15	RBF, gamma= 0.1, C= 100	9.022556	8.270677	17.293233	11.528822
16	RBF, gamma= 1, C= 0.1	45.112782	43.609023	49.624060	46.115288
17	RBF, gamma= 1, C= 1	28.571429	22.556391	41.353383	30.827068
18	RBF, gamma= 1, C= 10	27.819549	18.796992	37.593985	28.070175
19	RBF, gamma= 1, C= 100	27.819549	18.796992	37.593985	28.070175
20	RBF, gamma= 10, C= 0.1	45.112782	43.609023	49.624060	46.115288
21	RBF, gamma= 10, C= 1	45.112782	43.609023	49.624060	46.115288
22	RBF, gamma= 10, C= 10	45.112782	43.609023	49.624060	46.115288
23	RBF, gamma= 10, C= 100	45.112782	43.609023	49.624060	46.115288
24	RBF, gamma= 100, C= 0.1	45.112782	43.609023	49.624060	46.115288
25	RBF, gamma= 100, C= 1	45.112782	43.609023	49.624060	46.115288
26	RBF, gamma= 100, C= 10	45.112782	43.609023	49.624060	46.115288
27	RBF, gamma= 100, C= 100	45.112782	43.609023	49.624060	46.115288

5.0.2 Low Resolution Images

[22]:
```
misclassifications_lowres = [np.
 ↪concatenate([fold1_poly_misclassifications_lowres,fold1_RBF_misclassifications_lowres]),␣
 ↪np.
 ↪concatenate([fold2_poly_misclassifications_lowres,fold2_RBF_misclassifications_lowres]),␣
 ↪np.
 ↪concatenate([fold3_poly_misclassifications_lowres,fold3_RBF_misclassifications_lowres])]
kernel_summaries_lowres= [np.concatenate([fold1_poly_results_lowres,␣
 ↪fold1_RBF_results_lowres]), np.concatenate([fold2_poly_results_lowres,␣
 ↪fold2_RBF_results_lowres]), np.concatenate([fold3_poly_results_lowres,␣
 ↪fold3_RBF_results_lowres])]

low_res_summarydf = SVM_functions.get_optimum_params(misclassifications_lowres,␣
 ↪kernel_summaries_lowres)
```

```
Best average error:  [7.26817043 7.26817043]
Best parameters:  ['RBF, gamma= 0.1, C= 1', 'RBF, gamma= 0.1, C= 10']
```

[23]:
```
low_res_summarydf
```

[23]:

	Params	Fold1	Fold2	Fold3	Average
0	Polynomial, d= 1, C= 0.1	11.278195	6.766917	8.270677	8.771930
1	Polynomial, d= 1, C= 1	9.022556	11.278195	8.270677	9.523810
2	Polynomial, d= 1, C= 10	13.533835	15.037594	11.278195	13.283208
3	Polynomial, d= 1, C= 100	9.774436	14.285714	11.278195	11.779449

4	Polynomial, d= 2, C= 0.1	23.308271	20.300752	18.796992	20.802005
5	Polynomial, d= 2, C= 1	22.556391	20.300752	23.308271	22.055138
6	Polynomial, d= 2, C= 10	22.556391	20.300752	23.308271	22.055138
7	Polynomial, d= 2, C= 100	22.556391	20.300752	23.308271	22.055138
8	Polynomial, d= 3, C= 0.1	14.285714	7.518797	13.533835	11.779449
9	Polynomial, d= 3, C= 1	14.285714	7.518797	13.533835	11.779449
10	Polynomial, d= 3, C= 10	14.285714	7.518797	13.533835	11.779449
11	Polynomial, d= 3, C= 100	14.285714	7.518797	13.533835	11.779449
12	RBF, gamma= 0.1, C= 0.1	45.112782	43.609023	49.624060	46.115288
13	RBF, gamma= 0.1, C= 1	8.270677	6.015038	7.518797	7.268170
14	RBF, gamma= 0.1, C= 10	7.518797	6.766917	7.518797	7.268170
15	RBF, gamma= 0.1, C= 100	7.518797	7.518797	7.518797	7.518797
16	RBF, gamma= 1, C= 0.1	45.112782	43.609023	49.624060	46.115288
17	RBF, gamma= 1, C= 1	28.571429	22.556391	39.097744	30.075188
18	RBF, gamma= 1, C= 10	26.315789	21.804511	36.090226	28.070175
19	RBF, gamma= 1, C= 100	26.315789	21.804511	36.090226	28.070175
20	RBF, gamma= 10, C= 0.1	45.112782	43.609023	49.624060	46.115288
21	RBF, gamma= 10, C= 1	45.112782	43.609023	49.624060	46.115288
22	RBF, gamma= 10, C= 10	45.112782	43.609023	49.624060	46.115288
23	RBF, gamma= 10, C= 100	45.112782	43.609023	49.624060	46.115288
24	RBF, gamma= 100, C= 0.1	45.112782	43.609023	49.624060	46.115288
25	RBF, gamma= 100, C= 1	45.112782	43.609023	49.624060	46.115288
26	RBF, gamma= 100, C= 10	45.112782	43.609023	49.624060	46.115288
27	RBF, gamma= 100, C= 100	45.112782	43.609023	49.624060	46.115288

6 Using the SVM model trained on (opt, Copt), compute the classification error on the test set. This process must be repeated for each fold separately to compute the classification error for each fold as well as the average classification error over all folds as described earlier.

6.1 Fold 1

6.1.1 High Resolution

[24]:
```
test_eigencoefficients_f1 = np.loadtxt('GenderDataRowOrder/48_60/tsPCA_01.txt')
test_class_labels_f1  = np.loadtxt('GenderDataRowOrder/48_60/TtsPCA_01.txt')
```

[25]:
```
params= '-s 0 -t 2 -g 0.1 -c 1'
e1 = SVM_functions.run_SVM(train_eigencoefficients_f1[:,:30],␣
 ↪train_class_labels_f1, test_eigencoefficients_f1[:,:30],␣
 ↪test_class_labels_f1, params)
```

```
*.*
optimization finished, #iter = 165
nu = 0.565958
obj = -48.349285, rho = -0.448555
```

```
nSV = 96, nBSV = 55
Total nSV = 96
Accuracy = 80.4511% (107/133) (classification)
```

6.1.2 Low Resolution

[26]:
```
test_eigencoefficients_f1_lowres = np.loadtxt('GenderDataRowOrder/16_20/
  ↪tsPCA_01.txt')
test_class_labels_f1_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtsPCA_01.
  ↪txt')
```

[27]:
```
params= '-s 0 -t 2 -g 0.1 -c 1'
e1_lr = SVM_functions.run_SVM(train_eigencoefficients_f1_lowres[:,:30],␣
  ↪train_class_labels_f1_lowres, test_eigencoefficients_f1_lowres[:,:30],␣
  ↪test_class_labels_f1_lowres, params)
```

```
*.*
optimization finished, #iter = 168
nu = 0.610539
obj = -51.696565, rho = -0.202279
nSV = 107, nBSV = 62
Total nSV = 107
Accuracy = 90.9774% (121/133) (classification)
```

6.2 Fold 2

6.2.1 High Resolution

[28]:
```
test_eigencoefficients_f2 = np.loadtxt('GenderDataRowOrder/48_60/tsPCA_02.txt')
test_class_labels_f2  = np.loadtxt('GenderDataRowOrder/48_60/TtsPCA_02.txt')
```

[29]:
```
params= '-s 0 -t 2 -g 0.1 -c 1'
e2 = SVM_functions.run_SVM(train_eigencoefficients_f2[:,:30],␣
  ↪train_class_labels_f2, test_eigencoefficients_f2[:,:30],␣
  ↪test_class_labels_f2, params)
```

```
*.*
optimization finished, #iter = 172
nu = 0.617994
obj = -52.360434, rho = -0.111316
nSV = 105, nBSV = 57
Total nSV = 105
Accuracy = 91.7293% (122/133) (classification)
```

6.2.2 Low Resolution

[30]:
```
test_eigencoefficients_f2_lowres = np.loadtxt('GenderDataRowOrder/16_20/
↪tsPCA_02.txt')
test_class_labels_f2_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtsPCA_02.
↪txt')
```

[31]:
```
params= '-s 0 -t 2 -g 0.1 -c 1'
e2_lr = SVM_functions.run_SVM(train_eigencoefficients_f2_lowres[:,:30],␣
↪train_class_labels_f2_lowres, test_eigencoefficients_f2_lowres[:,:30],␣
↪test_class_labels_f2_lowres, params)
```

```
*.*
optimization finished, #iter = 174
nu = 0.653638
obj = -56.787041, rho = -0.268447
nSV = 108, nBSV = 63
Total nSV = 108
Accuracy = 90.9774% (121/133) (classification)
```

6.3 Fold 3

6.3.1 High Resolution

[32]:
```
test_eigencoefficients_f3 = np.loadtxt('GenderDataRowOrder/48_60/tsPCA_03.txt')
test_class_labels_f3  = np.loadtxt('GenderDataRowOrder/48_60/TtsPCA_03.txt')
```

[33]:
```
params= '-s 0 -t 2 -g 0.1 -c 1'
e3 = SVM_functions.run_SVM(train_eigencoefficients_f3[:,:30],␣
↪train_class_labels_f3, test_eigencoefficients_f3[:,:30],␣
↪test_class_labels_f3, params)
```

```
*.*
optimization finished, #iter = 135
nu = 0.543411
obj = -46.176618, rho = 0.346135
nSV = 91, nBSV = 54
Total nSV = 91
Accuracy = 86.4662% (115/133) (classification)
```

6.3.2 Low Resolution

[34]:
```
test_eigencoefficients_f3_lowres = np.loadtxt('GenderDataRowOrder/16_20/
↪tsPCA_03.txt')
test_class_labels_f3_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtsPCA_03.
↪txt')
```

[35]:
```
params= '-s 0 -t 2 -g 0.1 -c 1'
```

```
e3_lr= SVM_functions.run_SVM(train_eigencoefficients_f3_lowres[:,:30],␣
↪train_class_labels_f3_lowres, test_eigencoefficients_f3_lowres[:,:30],␣
↪test_class_labels_f3_lowres, params)
```

```
*.*
optimization finished, #iter = 167
nu = 0.614725
obj = -51.565169, rho = 0.069233
nSV = 102, nBSV = 58
Total nSV = 102
Accuracy = 90.2256% (120/133) (classification)
```

7 Misclassification rates for all folds

[36]:
```
print('Misclassification Error (48x60): \n \n', 'Fold 1:', e1, '\n Fold 2:',␣
↪e2, '\n Fold 3:', e3, '\n Average:', np.sum([e1,e2,e3])/3)
```

```
Misclassification Error (48x60):

 Fold 1: [19.54887218]
 Fold 2: [8.27067669]
 Fold 3: [13.53383459]
 Average: 13.78446115288221
```

[37]:
```
print('Misclassification Error (16x20): \n \n', 'Fold 1:', e1_lr, '\n Fold 2:',␣
↪e2_lr, '\n Fold 3:', e3_lr, '\n Average:', np.sum([e1_lr,e2_lr,e3_lr])/3)
```

```
Misclassification Error (16x20):

 Fold 1: [9.02255639]
 Fold 2: [9.02255639]
 Fold 3: [9.77443609]
 Average: 9.273182957393482
```

[ ]:
```

```

CS679_Assignment4_JaleesaHoule_Experiment2

May 3, 2024

[1]:
```
import numpy as np
import scipy
import BayesianClassifier as BC
import MaxLikelihoodEstimator as ML
```

1 Experiment 2: apply the Bayes classifier assuming a Gaussian distribution for each fold and use ML estimation to estimate the parameters for each class. Assume that the covariance matrix for each class is diagonal (set the off-diagonal elements to zero) and use the Mahalanobis distance for classification.

2 Fold 1

2.1 High Resolution

[2]:
```
# training data
train_eigencoefficients_f1  = np.loadtxt('GenderDataRowOrder/48_60/trPCA_01.
 ↪txt')
train_class_labels_f1  = np.loadtxt('GenderDataRowOrder/48_60/TtrPCA_01.txt')

# test data

test_eigencoefficients_f1 = np.loadtxt('GenderDataRowOrder/48_60/tsPCA_01.txt')
test_class_labels_f1  = np.loadtxt('GenderDataRowOrder/48_60/TtsPCA_01.txt')
```

2.1.1 Training

[3]:
```
f1 = ML.ML_Estimator() # initialize class instance
f1.samples = ML.sort_data(train_eigencoefficients_f1[:,:30],␣
 ↪train_class_labels_f1) #reorganize data
f1.priors= [0.5, 0.5] #set equal priors

f1.get_ML_param_estimates(corr=False, round_estimates = False) #estimate␣
 ↪covariance and mu values
f1.determine_case_and_params(param_estimates='ML')
```

2.1.2 Testing

[4]:
```
f1.samples= ML.sort_data(test_eigencoefficients_f1[:,:30],␣
 ↪test_class_labels_f1) ## feed in the test data
f1.convert_to_pandas_df()
f1.classify_by_mahalanobis_distance()
f1.get_error()
```

```
**********************************************************************


 Empirical Error Stats:

Percent of samples misclassified: 9.022556390977442

**********************************************************************
```

2.2 Low Resolution

[5]:
```
# training data
train_eigencoefficients_f1_lowres  = np.loadtxt('GenderDataRowOrder/16_20/
 ↪trPCA_01.txt')
train_class_labels_f1_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtrPCA_01.
 ↪txt')

# test data

test_eigencoefficients_f1_lowres = np.loadtxt('GenderDataRowOrder/16_20/
 ↪tsPCA_01.txt')
test_class_labels_f1_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtsPCA_01.
 ↪txt')
```

2.2.1 Training

[6]:
```
f1_low = ML.ML_Estimator() # initialize class instance
f1_low.samples = ML.sort_data(train_eigencoefficients_f1_lowres[:,:30],␣
 ↪train_class_labels_f1_lowres) #reorganize data
f1_low.priors= [0.5, 0.5] #set equal priors

f1_low.get_ML_param_estimates(corr=False, round_estimates = False) #estimate␣
 ↪covariance and mu values
f1_low.determine_case_and_params(param_estimates='ML')
```

2.2.2 Testing

[7]:
```
f1_low.samples= ML.sort_data(test_eigencoefficients_f1_lowres[:,:30],␣
 ↪test_class_labels_f1_lowres) ## feed in the test data
f1_low.convert_to_pandas_df()
f1_low.classify_by_mahalanobis_distance()
f1_low.get_error()
```

```
**********************************************************************


 Empirical Error Stats:

Percent of samples misclassified: 11.27819548872181

**********************************************************************
```

3 Fold 2

3.1 High Resolution

[8]:
```
# training data
train_eigencoefficients_f2  = np.loadtxt('GenderDataRowOrder/48_60/trPCA_02.
 ↪txt')
train_class_labels_f2  = np.loadtxt('GenderDataRowOrder/48_60/TtrPCA_02.txt')

# test data

test_eigencoefficients_f2 = np.loadtxt('GenderDataRowOrder/48_60/tsPCA_02.txt')
test_class_labels_f2  = np.loadtxt('GenderDataRowOrder/48_60/TtsPCA_02.txt')
```

3.1.1 Training

[9]:
```
f2 = ML.ML_Estimator() # initialize class instance
f2.samples = ML.sort_data(train_eigencoefficients_f2[:,:30],␣
 ↪train_class_labels_f2) #reorganize data
f2.priors= [0.5, 0.5] #set equal priors

f2.get_ML_param_estimates(corr=False, round_estimates = False) #estimate␣
 ↪covariance and mu values
f2.determine_case_and_params(param_estimates='ML')
```

3.1.2 Testing

[10]:
```
f2.samples= ML.sort_data(test_eigencoefficients_f2[:,:30],␣
 ↪test_class_labels_f2) ## feed in the test data
f2.convert_to_pandas_df()
f2.classify_by_mahalanobis_distance()
f2.get_error()
```

```
**********************************************************************


 Empirical Error Stats:

Percent of samples misclassified: 6.766917293233088

**********************************************************************
```

3.2 Low Resolution

[11]:
```
# training data
train_eigencoefficients_f2_lowres  = np.loadtxt('GenderDataRowOrder/16_20/
 ↪trPCA_02.txt')
train_class_labels_f2_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtrPCA_02.
 ↪txt')

# test data

test_eigencoefficients_f2_lowres = np.loadtxt('GenderDataRowOrder/16_20/
 ↪tsPCA_02.txt')
test_class_labels_f2_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtsPCA_02.
 ↪txt')
```

3.2.1 Training

[12]:
```
f2_low = ML.ML_Estimator() # initialize class instance
f2_low.samples = ML.sort_data(train_eigencoefficients_f2_lowres[:,:30],␣
 ↪train_class_labels_f2_lowres) #reorganize data
f2_low.priors= [0.5, 0.5] #set equal priors

f2_low.get_ML_param_estimates(corr=False, round_estimates = False) #estimate␣
 ↪covariance and mu values
f2_low.determine_case_and_params(param_estimates='ML')
```

3.2.2 Testing

[13]:
```
f2_low.samples= ML.sort_data(test_eigencoefficients_f2_lowres[:,:30],␣
 ↪test_class_labels_f2_lowres) ## feed in the test data
f2_low.convert_to_pandas_df()
f2_low.classify_by_mahalanobis_distance()
f2_low.get_error()
```

```
**********************************************************************


 Empirical Error Stats:

Percent of samples misclassified: 6.015037593984962

**********************************************************************
```

4 Fold 3

4.1 High Resolution

[14]:
```
# training data
train_eigencoefficients_f3  = np.loadtxt('GenderDataRowOrder/48_60/trPCA_03.
 ↪txt')
train_class_labels_f3  = np.loadtxt('GenderDataRowOrder/48_60/TtrPCA_03.txt')

# test data

test_eigencoefficients_f3 = np.loadtxt('GenderDataRowOrder/48_60/tsPCA_03.txt')
test_class_labels_f3  = np.loadtxt('GenderDataRowOrder/48_60/TtsPCA_03.txt')
```

4.1.1 Training

[15]:
```
f3 = ML.ML_Estimator() # initialize class instance
f3.samples = ML.sort_data(train_eigencoefficients_f3[:,:30],␣
 ↪train_class_labels_f3) #reorganize data
f3.priors= [0.5, 0.5] #set equal priors

f3.get_ML_param_estimates(corr=False, round_estimates = False) #estimate␣
 ↪covariance and mu values
f3.determine_case_and_params(param_estimates='ML')
```

4.1.2 Testing

[16]:
```
f3.samples= ML.sort_data(test_eigencoefficients_f3[:,:30],␣
 ↪test_class_labels_f3) ## feed in the test data
f3.convert_to_pandas_df()
f3.classify_by_mahalanobis_distance()
f3.get_error()
```

```
 ****************************************************************


 Empirical Error Stats:

Percent of samples misclassified: 11.27819548872181

 ****************************************************************
```

4.2 Low Resolution

[17]:
```
# training data
train_eigencoefficients_f3_lowres  = np.loadtxt('GenderDataRowOrder/16_20/
 ↪trPCA_03.txt')
train_class_labels_f3_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtrPCA_03.
 ↪txt')

# test data

test_eigencoefficients_f3_lowres = np.loadtxt('GenderDataRowOrder/16_20/
 ↪tsPCA_03.txt')
test_class_labels_f3_lowres  = np.loadtxt('GenderDataRowOrder/16_20/TtsPCA_03.
 ↪txt')
```

4.2.1 Training

[18]:
```
f3_low = ML.ML_Estimator() # initialize class instance
f3_low.samples = ML.sort_data(train_eigencoefficients_f3_lowres[:,:30],␣
 ↪train_class_labels_f3_lowres) #reorganize data
f3_low.priors= [0.5, 0.5] #set equal priors

f3_low.get_ML_param_estimates(corr=False, round_estimates = False) #estimate␣
 ↪covariance and mu values
f3_low.determine_case_and_params(param_estimates='ML')
```

4.2.2 Testing

[19]:
```
f3_low.samples= ML.sort_data(test_eigencoefficients_f3_lowres[:,:30],␣
 ↪test_class_labels_f3_lowres) ## feed in the test data
f3_low.convert_to_pandas_df()
f3_low.classify_by_mahalanobis_distance()
f3_low.get_error()
```

```
**********************************************************************


 Empirical Error Stats:

Percent of samples misclassified: 10.526315789473683

**********************************************************************
```

[ ]:
```

```