**CS488/588 Homework 3 -70 pts**

Due: March, 24 (Upload soft copy on Canvas as a single file with .docx or .pdf format) For each of the below questions provide your analysis/inference of the results.

1a. Write a python program for dimensionality reduction using PCA on the Iris and Indian Pines dataset that implements the following: (25 points)

1. For PCA, plot the explained variance for all the PC’s in the dataset.(5 points) ii) Reduced data visualization using PCA to 2 dimensions – display the new transformed data which is reduced to two dimensions for visualization. – display the first two PC’s (directions of projections) with respect to color-coded class separability. (10 points: 5 points per dataset for PCA plots, i.e. 5pts per plots)

A white rectangular frame with black squares

Description automatically generated

A blue rectangles with white background

Description automatically generated

A graph with green and orange dots

Description automatically generated

A graph showing a graph of colored dots

Description automatically generated with medium confidence

iii) Reduced data visualization using LDA to 2 dimensions – display the new transformed data which is reduced to two dimensions for visualization. – display the first two directions in LDA (directions of projections) with respect to color-coded class separability.

(10 points: 5 points per dataset for LDA plots, i.e. 5pts per plots) \*Note only provide data visualizations here.

A graph with a line in the middle

Description automatically generated with medium confidence

The number of dimensions chosen for analysis from i) can be anything – provide your justification in 1b for your choice. For visualization in ii) and iii) only plot the first two of dimensions chosen. Also, number of PCs or LDs chosen must be the same for all analyses.

**I chose to do one dimension because that’s kind of how LDA works. It uses a line to maximize the space between classes and minimize differences within classes.**

1b. Discuss the analysis of results from 1a) for each dataset in terms of:

Role of dimensionality reduction /feature extraction on data analysis, inferences about data separability and your choice of ‘K’ PC’s and LDA features to be retained in each dataset –which method worked best on each dataset and why?

(5 points: i.e. 2.5pts per dataset inferences)

\* Use data visualization to draw analysis

2a. Write a python program to perform supervised classification on the Iris and Indian Pines datasets using Naïve Bayes, and Support vector machines (with RBF and Poly kernel) classifiers for training sizes ={10%, 20%, 30%, 40%, 50%} for each of the below cases:

1. with dimensionality reduction – Reduce data based on your choice of ‘K’ dimensions from 1a) using each of the dimensionality reduction methods (PCA, LDA) followed by supervised classification by the listed classifiers.
2. without dimensionality reduction – data is followed by supervised classification using the listed classifiers.
3. Provide the plots for overall training accuracy, and overall classification accuracy vs. the training size for all methods (classification schemes). Tabulate the classwise classification accuracies (i.e. extension of the sensitivity and specificity values) only for 30% training size over all methods for each dataset for case i) i.e. with dimensionality reduction PCA and LDA for Indian pines dataset only.

(30 pts: Total 4 plots, i.e. 2 plots for each dataset [case i and ii, overall training, and testing accuracy] + 2 tables (classwise accuracy), i.e, 1 table per (PCA, LDA) = 30 pts total, i.e. 5 pts per plots/table)

\*Note only provide data visualizations here – label figures

Provide appropriate legends in all figures to denote the methods. For figures follow the below nomenclature: Figure 1: what it does the figure denote and for which dataset results and label all figures in the homework with a caption. (Eg. Figure 1. Classification accuracy with/without dimensionality reduction for Iris dataset.)

You can present PCA+ classification, LDA+ classification plots and without dimensionality reduction + classification plots separately.

2b. Discuss the analysis of results from 2a) for each dataset in terms of:

i) Role of dimensionality reduction /feature extraction on data analysis and classification performance based on data separability, classification accuracy, sensitivity and specificity parameters.—which supervised classification method worked best on each dataset either with or without dimensionality reduction casesand why?

**When performing data analysis, it is essential to get your data to a readable format so you can actually do the analysis. Since these sets have a lot of samples, it can be easy to get lost in all the raw data, hence, the reason why we perform strategies such as dimensionality reduction. Out of all the strategies, I would say that LDA worked the best only because each class is more separated.**

(10 points: i.e. 5pts per dataset inferences)

\* Use data visualization to draw analysis

A graph of different colored lines

Description automatically generated

Note: Clearly label each section of code and figures. For Indian Pines dataset, read the indianR.mat data given in the homework folder, where X-data, gth- groundtruth labels

Appendix

# Needed imports

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.io import loadmat

from sklearn.preprocessing import MinMaxScaler

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.pipeline import Pipeline

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split

from sklearn import datasets

from sklearn.metrics import confusion\_matrix

#######################################################################

# Load the data

df = loadmat('indianR.mat')

x = np.array(df['X'])

gth = np.array(df['gth'])

num\_rows = np.array(df['num\_rows'])

num\_cols = np.array(df['num\_cols'])

num\_bands = np.array(df['num\_bands'])

bands, samples = x.shape

# Load the ground truth data

gth\_mat = loadmat('indian\_gth.mat')

gth\_mat = {i : j for i, j in gth\_mat.items() if i[0] != '\_'}

gt = pd.DataFrame({i : pd.Series(j[0]) for i, j in gth\_mat.items()})

# List features

n = []

ind = []

for i in range(bands):

    n.append(i + 1)

for i in range(bands):

    ind.append('band' + str(n[i]))

features = ind

# Normalize the features (preprocessing)

# 'MinMaxScaler' is a form of normalization that scales the data to a range between 0 and 1

scaler\_model = MinMaxScaler()

scaler\_model.fit(x.astype(float))

x = scaler\_model.transform(x)

# Apply PCA to the normalized features

pca = PCA()

principalComponents = pca.fit\_transform(x)

# Display contribution of each principal component

ev = pca.explained\_variance\_ratio\_

# Dimensionality reduction via PCA

x1 = x.transpose()

X\_pca = np.matmul(x1, principalComponents)

X\_pca.shape

# Below is the data visualization (bar graph)

plt.bar(np.arange(1, len(ev) + 1), list(ev \* 100), label = 'Principal Components', color = 'b')

plt.legend()

# Label the x-axis and y-axis

plt.xlabel('Principal Components')

plt.ylabel('Variance Ratio')

# Label the x-axis with the principal components

pc = []

for i in range(len(ev)):

    pc.append('PC-' + str(i + 1))

plt.xticks(np.arange(1, len(ev) + 1), pc)

# Title and display the graph

plt.title('Variance Ratio of INDIAN PINES Dataset')

plt.plot

plt.show()

#######################################################################

# Load the IRIS dataset

iris = datasets.load\_iris()

X = iris.data

# Apply PCA to the IRIS dataset

pca = PCA()

pca.fit(X)

# Display the contribution of each principal component

ev = pca.explained\_variance\_ratio\_

plt.bar(np.arange(1, len(ev) + 1), list(ev \* 100), label = 'Principal Components', color = 'b')

plt.xlabel('Principal Components')

plt.ylabel('Variance Ratio')

# Label the x-axis with the principal components

pc = []

for i in range(len(ev)):

    pc.append('PC-' + str(i + 1))

# Label the x-axis

plt.xticks(np.arange(1, len(ev) + 1), pc)

# Title and display the graph

plt.showgrid = True

plt.title('Variance Ratio of IRIS Dataset')

plt.plot

plt.show()

#######################################################################

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Perform PCA

pca = PCA(n\_components=2)  # Reduce to 2 components

X\_pca = pca.fit\_transform(X)

# Plot the data points in the space defined by the first two principal components

for i, target\_name in enumerate(iris.target\_names):

    plt.scatter(X\_pca[y == i, 0], X\_pca[y == i, 1], label=target\_name)

plt.title('PCA of Iris dataset - First Two Principal Components')

plt.xlabel('PC-1')

plt.ylabel('PC-2')

plt.legend()

plt.grid(True)

plt.show()

#######################################################################

# Load the data

df = loadmat('indianR.mat')

x = np.array(df['X'])

gth = np.array(df['gth'])

num\_rows = np.array(df['num\_rows'])

num\_cols = np.array(df['num\_cols'])

num\_bands = np.array(df['num\_bands'])

bands, samples = x.shape

# Load the ground truth data

gth\_mat = loadmat('indian\_gth.mat')

gth\_mat = {i : j for i, j in gth\_mat.items() if i[0] != '\_'}

gt = pd.DataFrame({i : pd.Series(j[0]) for i, j in gth\_mat.items()})

# List features

n = []

ind = []

for i in range(bands):

    n.append(i + 1)

for i in range(bands):

    ind.append('band' + str(n[i]))

features = ind

# Normalize the features (preprocessing)

# 'MinMaxScaler' is a form of normalization that scales the data to a range between 0 and 1

scaler\_model = MinMaxScaler()

scaler\_model.fit(x.astype(float))

x = scaler\_model.transform(x)

# Apply PCA to the normalized features

pca = PCA(n\_components = 10)

principalComponents = pca.fit\_transform(x)

# Display contribution of each principal component

ev = pca.explained\_variance\_ratio\_

# Dimensionality reduction via PCA

x1 = x.transpose()

X\_pca = np.matmul(x1, principalComponents)

X\_pca.shape

# Model the dataframe

x\_pca\_df = pd.DataFrame(data = X\_pca, columns = ['PC-1', 'PC-2', 'PC-3', 'PC-4', 'PC-5', 'PC-6', 'PC-7', 'PC-8', 'PC-9', 'PC-10'])

# Add the labels

X\_pca\_df = pd.concat([x\_pca\_df, gt], axis = 1)

# Below is more visualization (scatter plot)

fig = plt.figure(figsize = (10, 10))

ax = fig.add\_subplot(1, 1, 1)

ax.set\_xlabel('PC-1', fontsize = 15)

ax.set\_ylabel('PC-2', fontsize = 15)

ax.set\_title('PCA on INDIAN PINES Dataset', fontsize = 20)

class\_num = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]

colors = ['r', 'g', 'b', 'y', 'm', 'c', 'k', 'r', 'g', 'b', 'y', 'm', 'c', 'k', 'b', 'r']

markerm = ['o', 'o', 'o', 'o', 'o', 'o', 'o', '+', '+', '+', '+', '+', '+', '+', '\*', '\*']

for target, color, m in zip(class\_num, colors, markerm):

    indicesToKeep = X\_pca\_df['gth'] == target

    ax.scatter(X\_pca\_df.loc[indicesToKeep, 'PC-1'], X\_pca\_df.loc[indicesToKeep, 'PC-2'], c = color, s = 9, marker = m)

ax.legend(class\_num)

ax.grid()

plt.show()

#######################################################################

# LDA on the IRIS dataset

iris = datasets.load\_iris()

X = iris.data

y = iris.target

target\_names = iris.target\_names

lda = LinearDiscriminantAnalysis(n\_components = 2)

X\_r2 = lda.fit(X, y).transform(X)

colors = ['navy', 'turquoise', 'darkorange']

plt.figure()

for color, i, target\_name in zip(colors, [0, 1, 2], target\_names):

    plt.scatter(X\_r2[y == i, 0], X\_r2[y == i, 0], alpha = .8, color = color, label = target\_name)

plt.legend(loc = 'best', shadow = False, scatterpoints = 1)

plt.title('LDA of IRIS dataset')

plt.show()

# Iris classification

iris = datasets.load\_iris()

X = iris.data

y = iris.target

target\_names = iris.target\_names

# Split the data into training and test sets

X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, y, test\_size = 0.20, random\_state = 1, shuffle = True)

def plot\_learning\_curve(classifier, X, y, steps = 10, train\_sizes = np.linspace(0.1, 1.0, 10), label = "", color = "r", axes = None):

    estimator = Pipeline([("scaler", MinMaxScaler()), ("classifier", classifier)])

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 101)

    train\_scores = []

    test\_scores = []

    train\_sizes = []

    for i in range(0, X\_train.shape[0], X\_train.shape[0] // steps):

        if (i == 0):

            continue

        X\_train\_i = X\_train[0 : i, :]

        y\_train\_i = y\_train[0 : i]

        estimator.fit(X\_train\_i, y\_train\_i)

        train\_scores.append(estimator.score(X\_train\_i, y\_train\_i) \* 100)

        test\_scores.append(estimator.score(X\_test, y\_test) \* 100)

        train\_sizes.append(i + 1)

    if (X\_train.shape[0] % steps != 0):

        estimator.fit(X\_train, y\_train)

        train\_scores.append(estimator.score(X\_train, y\_train) \* 100)

        test\_scores.append(estimator.score(X\_test, y\_test) \* 100)

        train\_sizes.append(X\_train.shape[0])

    if axes is None:

        \_, axes = plt.subplot(2)

    train\_s = np.linspace(10, 100, num = 5)

    axes[0].plot(train\_sizes, train\_scores, 'o-', color = color, label = label)

    axes[1].plot(train\_sizes, test\_scores, 'o-', color = color, label = label)

    print("Training Accuracy of", label, ": ", train\_scores[-1], "%")

    print("Testing Accuracy of", label, ": ", test\_scores[-1], "%")

    print("")

    return plt

# Create a model

\_, axes = plt.subplots(1, 2, figsize = (12, 5))

num\_steps = 10

classifier\_labels = {

                    "Logistic Regression": (LogisticRegression(max\_iter = 1000, random\_state = 1), "red"),

                    "Random Forest": (RandomForestClassifier(random\_state = 1), "green"),

                    "SVM = Linear": (SVC(kernel = 'linear', random\_state = 1), "blue"),

                    "SVM = RBF": (SVC(kernel = 'rbf', random\_state = 1), "yellow"),

                    "SVM = Poly": (SVC(kernel = 'poly', random\_state = 1), "orange"),

                    "kNN": (KNeighborsClassifier(n\_neighbors = 5), "purple"),

                    "Gaussian Naive Bayes": (GaussianNB(), "lime")

                    }

for label in classifier\_labels:

    classifier = classifier\_labels[label][0]

    color = classifier\_labels[label][1]

    plot\_learning\_curve(classifier, X, y, steps = num\_steps, color = color, label = label, axes = axes)

axes[0].set\_xlabel("% of Training examples")

axes[0].set\_ylabel("Overall Classification Accuracy")

axes[0].set\_title("Model Evaluation: IRIS dataset Training/Recall Accuracy")

axes[0].legend()

axes[1].set\_xlabel("% of Training examples")

axes[1].set\_ylabel("Training/Recall Accuracy")

axes[1].set\_title("Model Evaluation: Cross-Validation Accuracy")

axes[1].legend()

plt.show()

def plot\_per\_class\_accuracy(classifier, X, y, label, feature\_selection = None):

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.9, random\_state = 101)

    pipeline = Pipeline([("scaler", MinMaxScaler()), ("classifier", classifier)])

    pipeline.fit(X\_train, Y\_train)

    disp = confusion\_matrix(pipeline, X\_test, y\_test, cmap = plt.cm.Blues)

    plt.title(label)

    plt.show()

    true\_positive = disp.confusion\_matrix[1][1]

    false\_negative = disp.confusion\_matrix[1][0]

    print(label + " - Sensitivity: ", true\_positive / (true\_positive + false\_negative))

    print()

for label in classifier\_labels:

    classifier = classifier\_labels[label][0]

    plot\_per\_class\_accuracy(classifier, X, y, label)