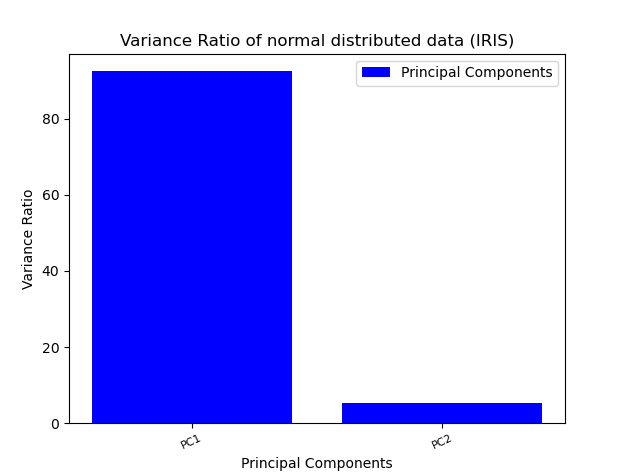
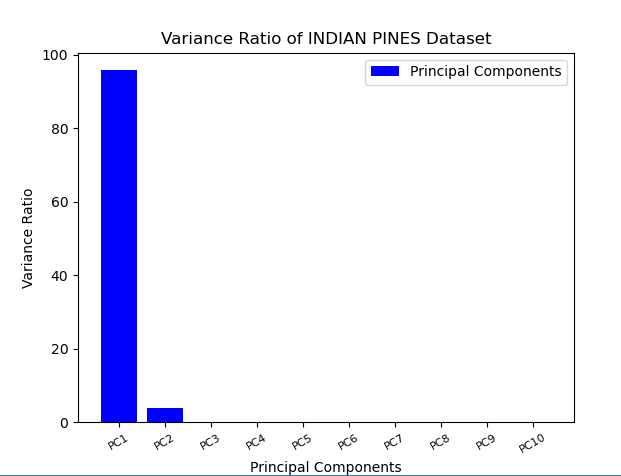
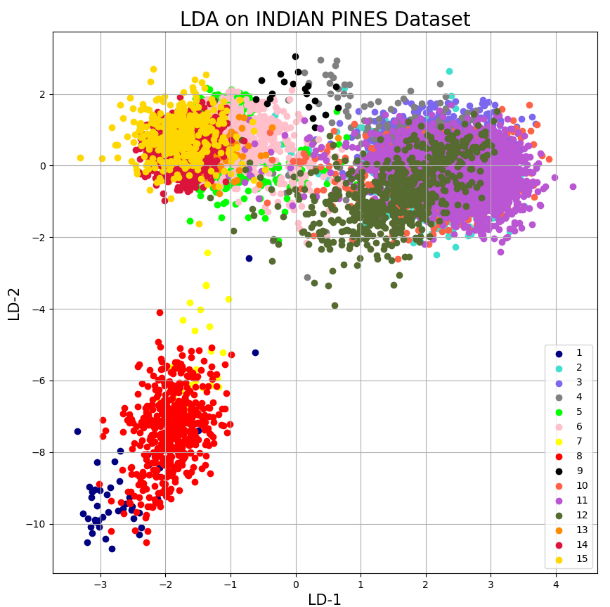
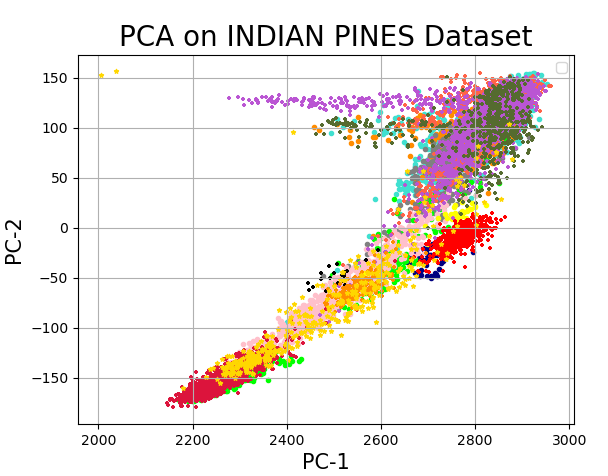
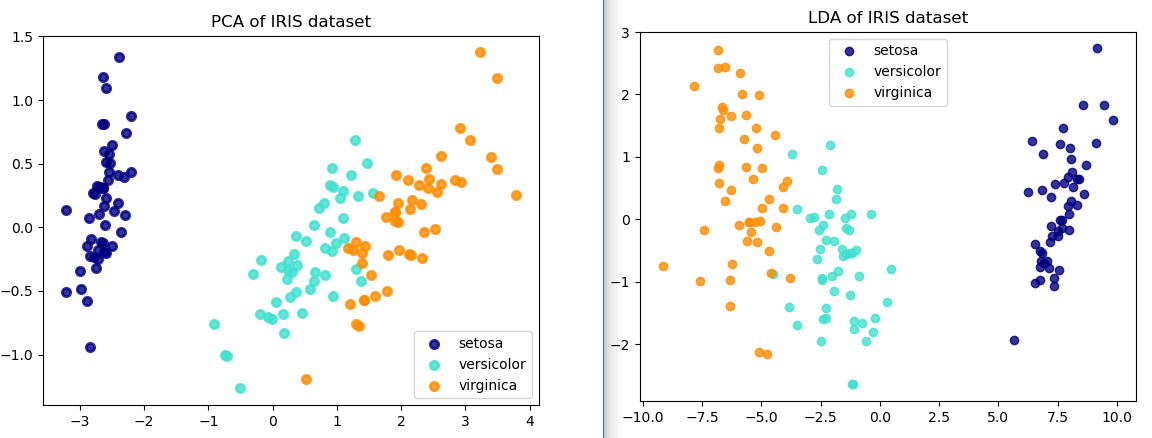
1a) Dimensionality reduction using PCA and LDA on the Iris and Indian Pines dataset

i) Figures 1 and 2 respectively: Plots of variance ratios of the Iris and Indian Pines dataset

ii) Figures 3, 4, 5, and 6 from top left to bottom right. Plots of 2D PCA and LDA dimensionality reduction on the Indian Pines and then Iris datasets





1b) Discuss the analysis of results

Pines Dataset:

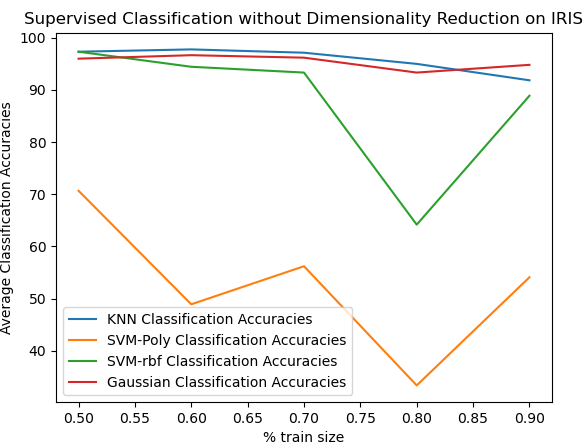
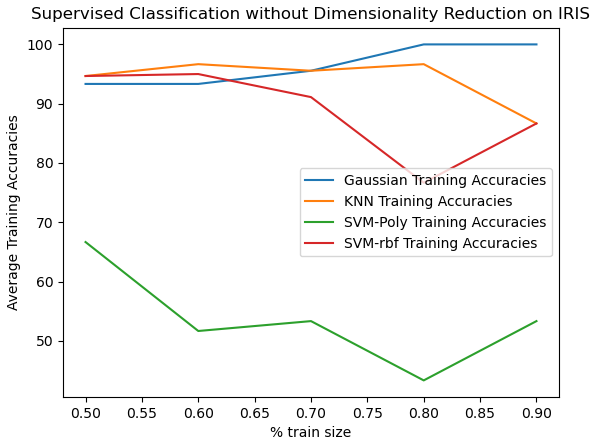
The HSI for the Indian Pines location had 202 layers of pixels for each pixel of ground truth, and only 150x150 pixels of ground truth. To a person, that's basically impossible to visualize the differences between each type of ground pixel. There's too many dimensions to the data to be understandable or visualizable. By performing dimensionality reduction and feature extraction we're able to see only the more important features and dimensions that are affecting the category a pixel is defined as. It unclutters the data so people don't have to see all the noise in the background and can instead view the distinction between classes based on the most important features. Reducing to only 2 dimensions here for display did overly restrict the distinction between classes, because there is a large amount of overlap in both the PCA and LDA plot. The classes have different trends, but it's difficult to separate which exactly is which, especially in the clumps of data. My takeaway here is that there is not a high level of data separability, because we were unable to cluster. For the Pines dataset I used K=10 for both PCA and LDA. That was the initial value in the demo video for running PCA, and it seemed to produce viable results so I kept it. The LDA seems to have clumped the different ground types better into clusters. I think because LDA is supervised, it was better able to handle the large amount of data and classifications. It's goal is to maximize feature separability, so it makes sense that it would separate the different classes into clusters better, whereas PCA might be a more accurate depiction of the similarities of the classes.

Iris Dataset:

The Iris dataset only has 4 features, so it's almost overkill to run dimensionality on it when you could probably classify the different classes by hand. However, it does enable you to display it in a very concise manner that's easy to understand. On the Iris dataset the clusters are almost entirely separate, so we can infer that there is high data separability and difference between the 3 classes. I chose K=2 for this dataset because that is the highest value allowed for LDA (the minimum of classes or features -1). I used that value for both PCA and LDA so there would be fewer differences in how they were run. It's hard to say which worked better on the Iris dataset because both gave very similar results. However, the LDA had fewer overlapping points between the Versicolor and Virginica so I would argue that it did better at dimensionality reduction.

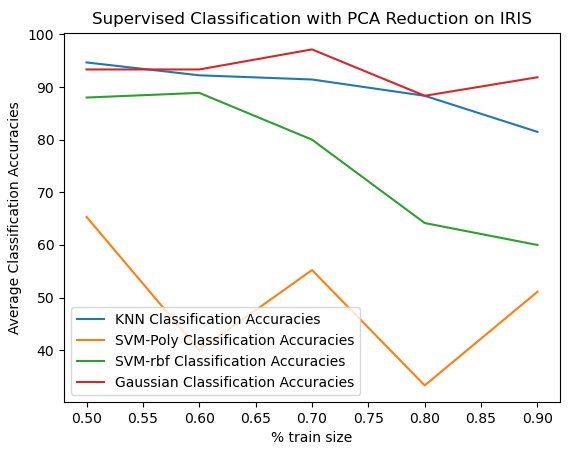
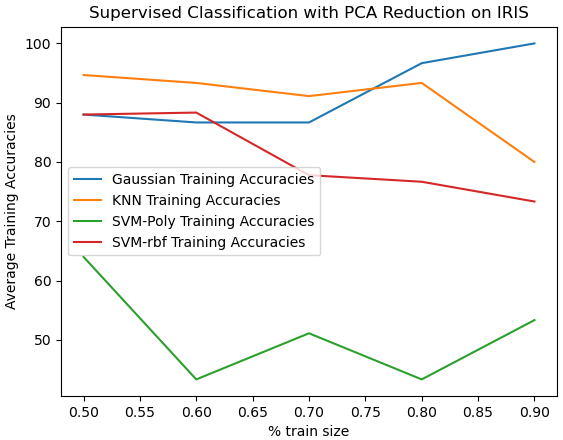
2a) Supervised classification on the Iris and Indian Pines datasets

IRIS) Figures 7,8,9,10 from top left to bottom right, plots of average model classification and training accuracy with respect to test size on the Iris dataset with no PCA/LDA, with PCA, with LDA



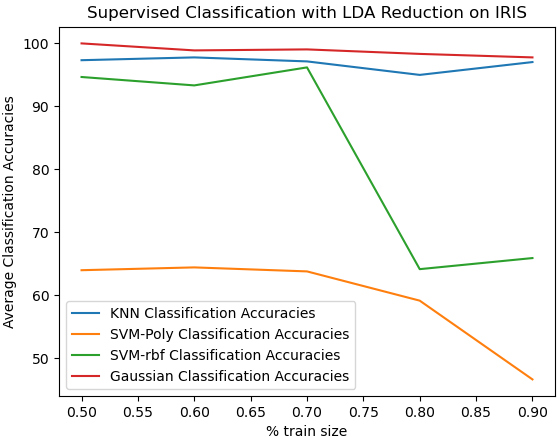
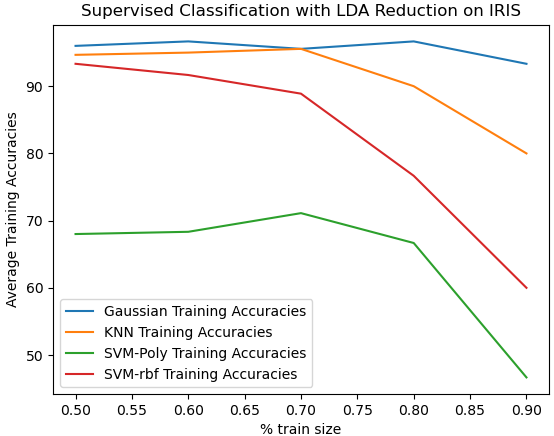
% test size

% test size



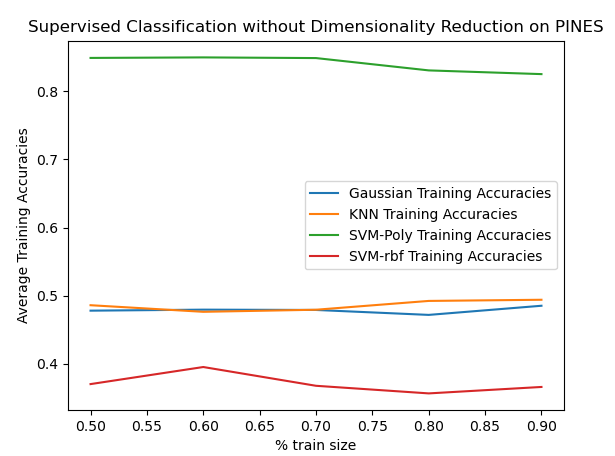
% test size

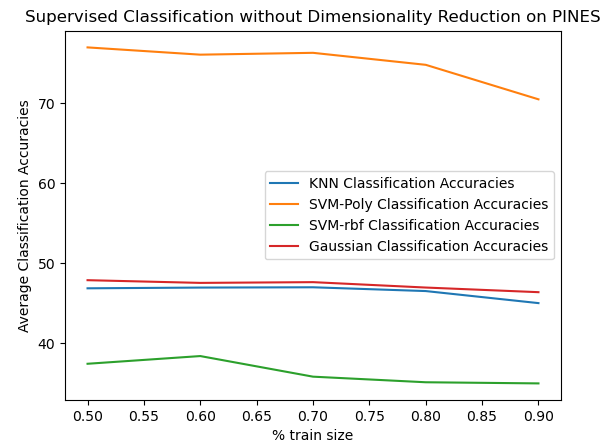
% test size



% test size

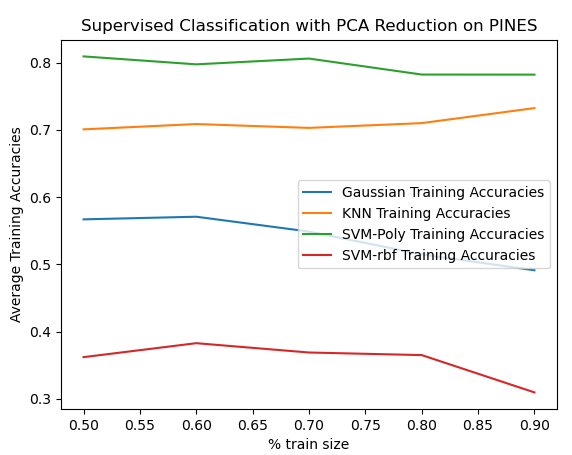
% test size

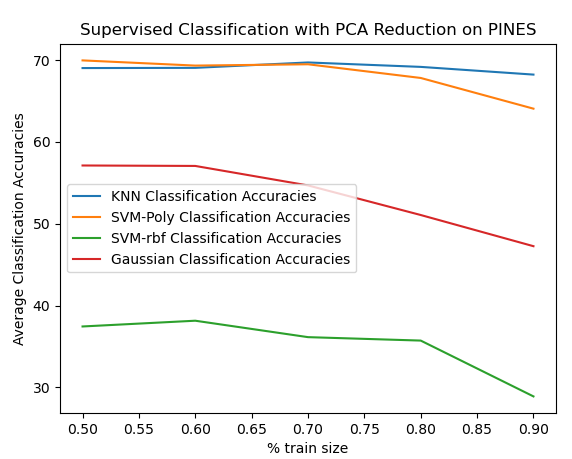
PINES) Figures 11,12,13,14 from top left to bottom right, plots of average model classification and training accuracy with respect to test size on the PINES dataset with no PCA/LDA, with PCA, with LDA



% test size

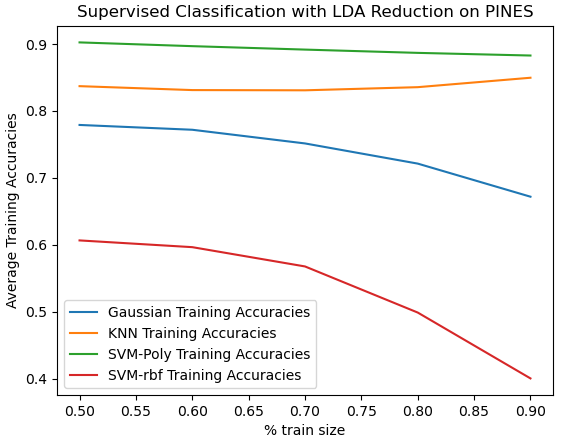
% test size

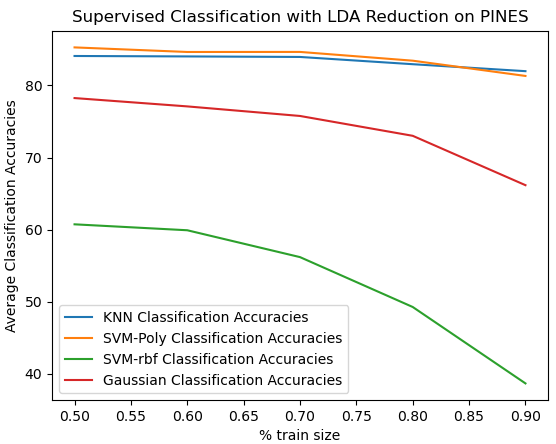




% test size

% test size





% test size

% test size

iii) Figure 15: tabularized class-wise classification accuracy charts for 30% training size over all methods without dimensionality reduction for the Iris dataset

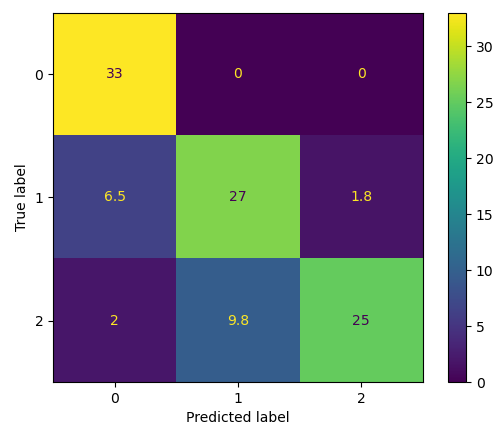
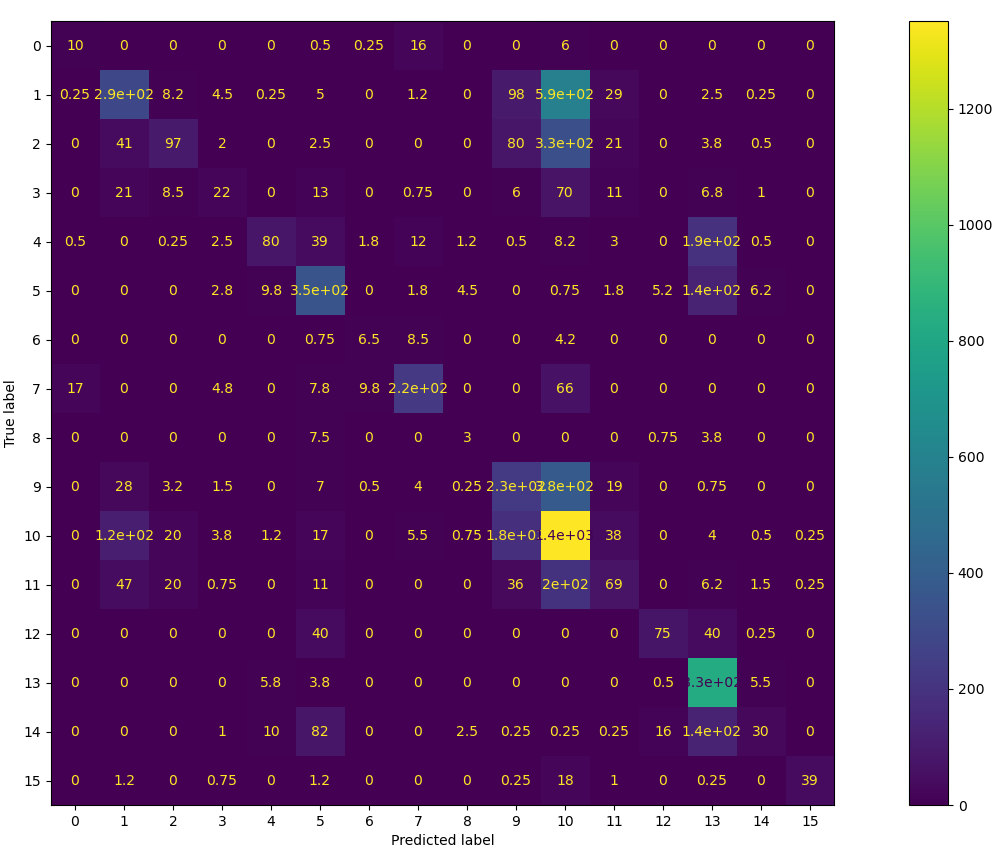


Figure 16: tabularized class-wise classification accuracy charts for 30% training size over all methods without dimensionality reduction for the Indian Pines dataset



2b) Discuss the analysis of results

Pines Dataset:

The highest achieved accuracy was with the SVM with a poly kernel on the dataset after using LDA. A very close second was the K-Nearest Neighbor after using LDA. K-Nearest Neighbor is a very strong model for machine learning, so I'm unsurprised that it performed well. However, because KNN does better with training data much larger than number of features, I think that as training size increases it might pull further ahead of the SVM (Varghese, 2018). It seems like there is enough data for the LDA to remove unimportant features and the polynomial kernel to simulate more effective ones and then train on that. The worst was the SVM with an rbf filter on every case. It's probable that if time were spent tuning the C and gamma parameters that SVM-rbf would have been more effective, as the parameters have a 'critical' effect on the model's performance according to sklearn's documentation. My quick research onto which kernel performs well on certain types of data did not return much. Many people stated that it's difficult to tell in advance and best to attempt multiple kernels to see which will work better for your specific problem.

Overall, the PCA and LDA had pretty drastic effects on the dataset. The 2 SVM's were relatively unaffected, and only increased or decreased within 10% accuracy after LDA or PCA was run, but KNN and Gaussian jumped at least 30% accuracy after LDA was run. However, LDA had a higher accuracy for every model. This means the earlier hypothesis that LDA was more effective on the Pines dataset was correct. The reason the LDA and PCA improve performance so much must be that there are more dimensions than are useful for a model on this size of data. Reducing the dimensions allows the models to pick up on more important features that correspond with classes. This makes sense because there are 202 layers for each pixel, but we found earlier that only 2 PC's had a strong correlation. Without PCA or LDA, 3 of the models didn't even improve more than 5% even as the training size increased from 10% to 50% of the data. With the LDA, there are distinct curves of improvement for both the SVM with rbf and Gaussian models.

Although the plots are not displayed, LDA and PCA also improve the confusion matrices created with a train size of 30%. The main diagonal skews lighter and there are fewer bright spots outside of correct choices. This implies that dimensionality reduction also has a positive effect on the sensitivity and specificity of a model if the dataset has too many dimensions and features compared to size.

Iris Dataset:

A lot of the discussion of the Iris dataset analysis hinges on the fact that it is a very small dataset with very few features, which makes it prone to overfitting. Running dimensionality reduction is useful if there is too much noise and a model will struggle to pick up on the important features. With a dataset that only has 4 features, dimensionality reduction will take away important and useful features, making it more difficult for supervised models to learn. Because of this, running the PCA before training models decreased model classification and training accuracy or left it the same in every single case on the 30% test size, and in most averages of all the models over each test size. Sensitivity and specificity were again not affected by running PCA. Another interesting point is the overfitting. The SVM with a poly kernel severely overfits the data at 30% test size, with a 94% training accuracy and a 51-55% validation accuracy. It's higher after running PCA. The SVM with an rbf kernel overfits as well but to less of an extreme. You can see it as well in the average accuracies from 20-40% training sizes, where the training accuracy is about 10% above the classification accuracy. These solutions might be overly complex for such a simple dataset. The K-Nearest Neighbors once again performs the best on the 30% training size, winning by 2% over the SVM with rbf kernel on PCA data and the Gaussian model on both the PCA and raw data. I was surprised by this because I figured for a simpler dataset the SVM's would work better. The RBF kernel was close without PCA, so the kernel must have a large effect on how an SVM works. Perhaps a simpler kernel like linear would produce better results. The Gaussian model was also a high performer, which makes sense with what I said earlier on the Pines dataset, that it would work better on a more evenly distributed dataset. Overall, it seems like the K-Nearest Neighbors model is the most effective model for datasets in this range.

Varghese: https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222

Code Appendix:

import scipy.io as io

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split, StratifiedKFold, cross\_val\_score

from sklearn import datasets

from sklearn.decomposition import PCA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.preprocessing import MinMaxScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import plot\_confusion\_matrix, ConfusionMatrixDisplay

from sklearn.pipeline import Pipeline

def pines\_analysis(R\_filepath, display=False):

    # Load in Indian Pines data from file and reshape/make into dataframes

    R\_file = io.loadmat(R\_filepath)

    gth = np.array(R\_file['gth'])

    X = np.array(R\_file['X'])

    R\_rows = R\_file['num\_rows']

    R\_cols = R\_file['num\_cols']

    R\_bands = R\_file['num\_bands']

    # Store ground truth data

    gth = np.reshape(gth, (int(R\_rows)\*int(R\_cols)))

    gth\_mat = io.loadmat('data/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()})

    # Pre-process data

    scaler\_model = MinMaxScaler()

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

    X = scaler\_model.transform(X)

    # Give a heads up to the user about data information

    print('gt shape:', gt.shape)

    print('X shape:', X.shape)

    # Class types in order of the Pines dataset

    target\_names = ["Alfalfa", "Corn-notill", "Corn-mintill", "Corn", "Grass-pasture",

                    "Grass-trees", "Grass-pasture-mowed", "Hay-windrowed", "Oats",

                    "Soybean-notill", "Soybean-mintill", "Soybean-clean", "Wheat", "Woods",

                    "Buildings-Grass-Treess-Drives", "Stone-Steel-Towers"]

    # Starting PCA

    pca = PCA(n\_components=10)

    principleComponents = pca.fit\_transform(X)

    # Creating dataframes from PCA output

    principleDf = pd.DataFrame(data=principleComponents,

                               columns = ['PC-' + str(i+1) for i in range(10)])

    finalDf1 = pd.concat([principleDf, gt], axis=1)

    # Reshaping PCA output for plotting, adding PC titles to each column

    x1 = X.transpose()

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

    x\_pca\_df = pd.DataFrame(data=X\_pca, columns=['PC-' + str(i+1) for i in range(10)])

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

    # Starting LDA

    X = X.transpose()

    lda = LinearDiscriminantAnalysis(n\_components=10)

    linear\_discriminants = lda.fit(X, np.ravel(gt)).transform(X)

    # Making dataframes from LDA

    linearDf = pd.DataFrame(data=linear\_discriminants,

                            columns = ['LD-' + str(i+1) for i in range(10)])

    finalDf2 = pd.concat([linearDf, gt], axis=1)

    # Reshaping output of LDA so it can be plotted, assigning LD's to each column of data

    x2 = X.transpose()

    X\_lda = np.matmul(x2, linear\_discriminants)

    x\_lda\_df = pd.DataFrame(data=X\_lda, columns=['LD-' + str(i+1) for i in range(10)])

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

    if display:

        # Lists that will be re-used for each plot

        class\_num = [i+1 for i in range(15)]

        colors = ["navy", "turquoise", "mediumslateblue", "gray", "lime", "pink", "yellow",

                  "red", "black", "tomato", "mediumorchid", "darkolivegreen", "darkorange",

                  "crimson", "gold", "peru", "mediumslateblue"]

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

        # Displaying Variance Ratio

        plt.figure()

        plt.bar([1,2,3,4,5,6,7,8,9,10], list(pca.explained\_variance\_ratio\_\*100),label="Principal Components", color="b")

        plt.legend()

        plt.xlabel('Principal Components')

        pc = []

        for i in range(10):

            pc.append('PC' + str(i+1))

        plt.xticks([1,2,3,4,5,6,7,8,9,10],pc,fontsize=8,rotation=30)

        plt.ylabel('Variance Ratio')

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

        plt.show()

        # Displaying PCA

        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)

        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, marker=m, s=9)

        ax.legend()

        ax.grid()

        plt.show()

        # Displaying LDA

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

        ax = fig.add\_subplot(1,1,1)

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

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

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

        for color, i, target\_name in zip(colors, class\_num, class\_num):

            ax.scatter(linear\_discriminants[gth==i, 0], linear\_discriminants[gth==i,1],

                        color=color, label=target\_name)

        ax.legend()

        ax.grid()

        plt.show()

def iris\_analysis(display=False):

    # Load iris dataset into data and targets

    iris = datasets.load\_iris()

    X = iris.data

    y = iris.target

    target\_names = iris.target\_names

    # Run PCA on the iris dataset

    pca = PCA(n\_components=2)

    X\_r = pca.fit(X).transform(X)

    # Run LDA on the iris dataset

    lda = LinearDiscriminantAnalysis(n\_components=2)

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

    # Percentage of variance explained for each components

    print(

        "explained variance ratio (first two components): %s"

        % str(pca.explained\_variance\_ratio\_)

    )

    # Plot figures if prompted

    if display:

        # Show the first 2 PC's

        plt.figure()

        plt.bar([1,2],list(pca.explained\_variance\_ratio\_\*100), label='Principal Components', color='b')

        plt.legend()

        plt.xlabel("Principal Components")

        pc = []

        for i in range(2):

            pc.append('PC' + str(i+1))

        plt.xticks([1,2],pc,fontsize=8,rotation=25)

        plt.ylabel("Variance Ratio")

        plt.title("Variance Ratio of normal distributed data (IRIS)")

        # Display 2D PCA results

        plt.figure()

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

        lw = 2

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

            # Show the PCA results for each datatype in the same color

            plt.scatter(

                X\_r[y == i, 0], X\_r[y == i, 1], color=color, alpha=0.8, lw=lw, label=target\_name

            )

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

        plt.title("PCA of IRIS dataset")

        # Display 2D LDA results

        plt.figure()

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

            # Show the LDA results for each datatype in the same color

            plt.scatter(

                X\_r2[y == i, 0], X\_r2[y == i, 1], alpha=0.8, color=color, label=target\_name

            )

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

        plt.title("LDA of IRIS dataset")

        # Show all the plots to screen

        plt.show()

def iris\_classification(run\_pca=False, run\_lda=True):

    if run\_pca and run\_lda:

        print("Can't run both lda and pca.")

        return

    # Load IRIS dataset, run PCA, and run combinations of all test sizes and model types

    # Display output to the user's screen

    # Load iris dataset into X and y variables

    iris = datasets.load\_iris()

    X = iris.data

    y = iris.target

    plot\_data = {}

    # Check for PCA flag and run dimensionality reduction if needed

    if run\_pca:

        pca = PCA(n\_components=2)

        X = pca.fit\_transform(X)

    if run\_lda:

        lda = LinearDiscriminantAnalysis(n\_components=2)

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

    test\_sizes = [.9, .8, .7, .6, .5]

    confusion\_matrices = []

    for test\_size in test\_sizes:

        # Split dataset based on test size

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

        # Re-make each model for the current test size

        models = [('KNN', KNeighborsClassifier()), ('SVM-Poly', SVC(gamma='auto', kernel='poly')),

                  ('SVM-RBF', SVC(gamma='auto', kernel='rbf')), ('NB', GaussianNB())]

        # Lists to hold average accuracies for later

        classification\_accuracy = []

        training\_accuracy = []

        for name, model in models:

            # Create a pipeline to preprocess data and run the model

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

            # Fit to the training dataset

            pipeline.fit(X\_train, Y\_train)

            # Create a confusion matrix

            if test\_size == .7:

                disp = plot\_confusion\_matrix(pipeline, X\_validation, Y\_validation, cmap=plt.cm.Blues)

                confusion\_matrices.append(disp.confusion\_matrix)

            # Calculate and store overall accuracy of the model

            classification\_score = pipeline.score(X\_validation, Y\_validation)

            classification\_accuracy.append(classification\_score\*100)

            # Calculate and store training accuracy of the model

            training\_accuracy.append(pipeline.score(X\_train, Y\_train)\*100)

            # Print model information to display

            data\_line = "test size: " + str(test\_size) + " model: " + str(name) + " training accuracy: " +\

                        str(pipeline.score(X\_train, Y\_train)\*100) + " classification accuracy: " +\

                        str(classification\_score\*100)

            print(data\_line)

        # Combine all the averages per test size

        plot\_data[test\_size] = [training\_accuracy, classification\_accuracy]

    confusion\_matrices = np.array(confusion\_matrices)

    avg\_matrix = np.mean(confusion\_matrices, axis=0)

    print("Confusion matrix:", avg\_matrix)

    plt.figure()

    display\_matrix = ConfusionMatrixDisplay(avg\_matrix)

    display\_matrix.plot()

    plt.show()

    # Display average overall accuracy and training accuracy to output for each test size

    training\_accuracies = [plot\_data[x][0] for x in plot\_data]

    print("Training accuracies:", training\_accuracies)

    classification\_accuracies = [plot\_data[x][1] for x in plot\_data]

    print("Classification accuracies:", classification\_accuracies)

def pines\_classification(R\_filepath, run\_pca=False, run\_lda=False):

    if run\_pca and run\_lda:

        print("Can't run both pca and lda")

        return

    # Runs combinations of model types/test sizes with or without PCA dimensionality reduction on pines dataset

    # Loading in the pines dataset

    R\_file = io.loadmat(R\_filepath)

    gth = np.array(R\_file['gth'])

    X = np.array(R\_file['X']).transpose()

    R\_rows = R\_file['num\_rows']

    R\_cols = R\_file['num\_cols']

    R\_bands = R\_file['num\_bands']

    gth = np.reshape(gth, (int(R\_rows)\*int(R\_cols)))

    # Ugly but works. Remove all the data where there isn't a ground truth

    zeros = (gth == 0).nonzero()

    print(gth.shape)

    print(X.shape)

    print("zeros:", zeros)

    for i in range(len(zeros)):

        zeros\_new = (gth == 0).nonzero()

        delete\_index = zeros\_new[0]

        gth = np.delete(gth, delete\_index)

        X = np.delete(X, delete\_index, axis=0)

    print(gth.shape)

    print(X.shape)

    if run\_pca:

        # Run PCA on data and save back to the X variable

        pca = PCA(n\_components=10)

        principleComponents = pca.fit\_transform(X)

        x1 = X.transpose()

        X = np.matmul(x1, principleComponents)

        print("X:", principleComponents.shape)

        print("gth:", gth.shape)

        X = principleComponents

        print("Running with PCA dimensionality reduction")

    elif run\_lda:

        lda = LinearDiscriminantAnalysis(n\_components=10)

        linear\_discriminants = lda.fit(X, np.ravel(gth)).transform(X)

        X = linear\_discriminants

        print("X:", X.shape)

        print("gth:", gth.shape)

        print("Running with LDA reduction")

    else:

        # Display data info

        print("X:", X.shape)

        print("gth:", gth.shape)

        print("Running without dimensionality reduction")

    print("GTH max class:", gth.max())

    plot\_data = {}

    test\_sizes = [.9, .8, .7, .6, .5]

    confusion\_matrices = []

    for test\_size in test\_sizes:

        # Splitting data by test size

        X\_train, X\_validation, Y\_train, Y\_validation = train\_test\_split(X, gth, test\_size=test\_size,

                                                                        random\_state=1, shuffle=True)

        # Re-making each model to use on the current test size (cache size taken from CPU cache size to go faster)

        models = [('NB', GaussianNB()), ('KNN', KNeighborsClassifier()), ('SVM-Poly', SVC(gamma='auto', kernel='poly', cache\_size=5200000)),

                  ('SVM-RBF', SVC(gamma='auto', kernel='rbf', cache\_size=5200000))]

        classification\_accuracy = []

        training\_accuracy = []

        for name, model in models:

            # Create a pipeline to preprocess data and run the model

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

            # Fit to the training dataset

            pipeline.fit(X\_train, Y\_train)

            if test\_size == .7:

                # Make a confusion matrix

                disp = plot\_confusion\_matrix(pipeline, X\_validation, Y\_validation, cmap=plt.cm.Blues)

                confusion\_matrices.append(disp.confusion\_matrix)

            # Overall model classification score vs training data score

            classification\_score = pipeline.score(X\_validation, Y\_validation)

            train\_score = pipeline.score(X\_train, Y\_train)

            # Add to a list to calculate the average later

            training\_accuracy.append(train\_score)

            classification\_accuracy.append(classification\_score\*100)

            # Printing model information to output

            data\_line = "test size: " + str(test\_size) + " model: " + str(name) + " training accuracy: " + str(train\_score\*100) + " classification accuracy: " + str(classification\_score\*100)

            print(data\_line)

        # Adding all model averages to a dictionary corresponding with test size

        plot\_data[test\_size] = [training\_accuracy, classification\_accuracy]

    # Average out all the display matrices and plot it

    confusion\_matrices = np.array(confusion\_matrices)

    avg\_matrix = np.mean(confusion\_matrices, axis=0)

    print("Confusion matrix:", avg\_matrix)

    plt.figure()

    display\_matrix = ConfusionMatrixDisplay(avg\_matrix)

    display\_matrix.plot()

    plt.show()

    # Displaying overall accuracies for each test size

    test\_sizes = [x\*100 for x in test\_sizes]

    training\_accuracies = [plot\_data[x][0] for x in plot\_data]

    print("Training accuracies:", training\_accuracies)

    classification\_accuracies = [plot\_data[x][1] for x in plot\_data]

    print("Classification accuracies:", classification\_accuracies)

def plot\_classification(iris, pines):

    # Plots the training/overall accuracies of each model at each test-size

    test\_sizes = [.9, .8, .7, .6, .5]

    if pines:

        # Saved data from PINES classification output (Hard-coded to avoid re-running models)

        # No reduction, PCA, LDA

        classification\_accuracies = [[[45.018970189701896, 70.47154471544715, 34.99186991869919, 46.38482384823848], [46.52439024390244, 74.78048780487805, 35.13414634146341, 46.96341463414634], [46.99651567944251, 76.2787456445993, 35.83275261324042, 47.63763066202091], [46.959349593495936, 76.04878048780488, 38.40650406504065, 47.54471544715447], [46.868292682926835, 76.9560975609756, 37.44390243902439, 47.88292682926829]],

                                     [[68.21680216802169, 64.05420054200542, 28.899728997289976, 47.2520325203252], [69.15853658536585, 67.8048780487805, 35.71951219512195, 51.0609756097561], [69.70034843205575, 69.4773519163763, 36.139372822299656, 54.67595818815331], [69.04065040650407, 69.3170731707317, 38.146341463414636, 57.05691056910569], [69.01463414634146, 69.95121951219512, 37.44390243902439, 57.11219512195122]],

                                     [[81.97289972899729, 81.31165311653118, 38.68834688346883, 66.16802168021681], [82.9390243902439, 83.42682926829268, 49.28048780487805, 73.01219512195122], [83.94425087108014, 84.6411149825784, 56.19512195121952, 75.76306620209058], [84.01626016260163, 84.6341463414634, 59.91869918699187, 77.08943089430895], [84.0780487804878, 85.26829268292683, 60.74146341463415, 78.2439024390244]]]

        training\_accuracies = [[[0.494140625, 0.8251953125, 0.3662109375, 0.4853515625], [0.4924353343094192, 0.8306490971205466, 0.35675939482674474, 0.4719375305026842], [0.47950553025374104, 0.8487312947299935, 0.36792452830188677, 0.47918022121014964], [0.4764576726030739, 0.8497194437667723, 0.3954623078799707, 0.47962917784825565], [0.48614363778298203, 0.8489461358313818, 0.3704137392661983, 0.4781420765027322]],

                               [[0.732421875, 0.7822265625, 0.3095703125, 0.4912109375], [0.7101024890190337, 0.7823328452903856, 0.36505612493899464, 0.5148853099072719], [0.7029928432010409, 0.8061158100195186, 0.36890045543266103, 0.5487963565387117], [0.708709441327153, 0.797511588192242, 0.3827762868992437, 0.5708709441327153], [0.7008196721311475, 0.8093286494925839, 0.3620218579234973, 0.5669398907103825]],

                               [[0.849609375, 0.8828125, 0.400390625, 0.671875], [0.8355295265983407, 0.8867740361151781, 0.4987798926305515, 0.72132747681796], [0.8308392973324659, 0.8916720884840599, 0.5676642810670136, 0.7514638906961614], [0.8311783361795559, 0.8968040985606246, 0.5964869480361064, 0.771895584288851], [0.8370413739266198, 0.9024199843871975, 0.6065573770491803, 0.7790788446526151]]]

        for i, accuracies in \

                enumerate(zip(classification\_accuracies, training\_accuracies)):

            classification\_accuracies\_itr = accuracies[0]

            training\_accuracies\_itr = accuracies[1]

            KNN\_accuracies = [x[0] for x in classification\_accuracies\_itr]

            KNN\_train\_accuracies = [x[0] for x in training\_accuracies\_itr]

            SVM\_poly\_accuracies = [x[1] for x in classification\_accuracies\_itr]

            SVM\_poly\_train\_accuracies = [x[1] for x in training\_accuracies\_itr]

            SVM\_rbf\_accuracies = [x[2] for x in classification\_accuracies\_itr]

            SVM\_rbf\_train\_accuracies = [x[2] for x in training\_accuracies\_itr]

            GNB\_accuracies = [x[3] for x in classification\_accuracies\_itr]

            GNB\_train\_accuracies = [x[3] for x in training\_accuracies\_itr]

            plt.plot(test\_sizes, KNN\_accuracies, label="KNN Classification Accuracies")

            plt.plot(test\_sizes, SVM\_poly\_accuracies,label="SVM-Poly Classification Accuracies")

            plt.plot(test\_sizes, SVM\_rbf\_accuracies,label="SVM-rbf Classification Accuracies")

            plt.plot(test\_sizes, GNB\_accuracies,label="Gaussian Classification Accuracies")

            plt.xlabel("% test size")

            plt.ylabel("Average Classification Accuracies")

            if i == 0:

                plt.title("Supervised Classification without Dimensionality Reduction on PINES")

            elif i == 1:

                plt.title("Supervised Classification with PCA Reduction on PINES")

            elif i == 2:

                plt.title("Supervised Classification with LDA Reduction on PINES")

            plt.legend()

            plt.show()

            plt.figure()

            plt.plot(test\_sizes, GNB\_train\_accuracies, label="Gaussian Training Accuracies")

            plt.plot(test\_sizes, KNN\_train\_accuracies, label="KNN Training Accuracies")

            plt.plot(test\_sizes, SVM\_poly\_train\_accuracies, label="SVM-Poly Training Accuracies")

            plt.plot(test\_sizes, SVM\_rbf\_train\_accuracies, label="SVM-rbf Training Accuracies")

            plt.xlabel("% test size")

            plt.ylabel("Average Training Accuracies")

            if i == 0:

                plt.title("Supervised Classification without Dimensionality Reduction on PINES")

            elif i == 1:

                plt.title("Supervised Classification with PCA Reduction on PINES")

            elif i == 2:

                plt.title("Supervised Classification with LDA Reduction on PINES")

            plt.legend()

            plt.show()

    if iris:

        # Saved data from IRIS classification output (Hard-coded to avoid re-running models)

        # No reduction, PCA, LDA <- data order

        training\_accuracies = [[[86.66666666666667, 53.333333333333336, 86.66666666666667, 100.0],

                               [96.66666666666667, 43.333333333333336, 76.66666666666667, 100.0],

                               [95.55555555555556, 53.333333333333336, 91.11111111111111, 95.55555555555556],

                               [96.66666666666667, 51.66666666666667, 95.0, 93.33333333333333],

                               [94.66666666666667, 66.66666666666666, 94.66666666666667, 93.33333333333333]],

                               [[80.0, 53.333333333333336, 73.33333333333333, 100.0],

                                [93.33333333333333, 43.333333333333336, 76.66666666666667, 96.66666666666667],

                                [91.11111111111111, 51.11111111111111, 77.77777777777779, 86.66666666666667],

                                [93.33333333333333, 43.333333333333336, 88.33333333333333, 86.66666666666667],

                                [94.66666666666667, 64.0, 88.0, 88.0]],

                               [[80.0, 46.666666666666664, 60.0, 93.33333333333333],

                                [90.0, 66.66666666666666, 76.66666666666667, 96.66666666666667],

                                [95.55555555555556, 71.11111111111111, 88.88888888888889, 95.55555555555556],

                                [95.0, 68.33333333333333, 91.66666666666666, 96.66666666666667],

                                [94.66666666666667, 68.0, 93.33333333333333, 96.0]]]

        classification\_accuracies = [[[91.85185185185185, 54.074074074074076, 88.88888888888889, 94.81481481481482],

                                     [95.0, 33.33333333333333, 64.16666666666667, 93.33333333333333],

                                     [97.14285714285714, 56.19047619047619, 93.33333333333333, 96.19047619047619],

                                     [97.77777777777777, 48.888888888888886, 94.44444444444444, 96.66666666666667],

                                     [97.33333333333334, 70.66666666666667, 97.33333333333334, 96.0]],

                                     [[81.48148148148148, 51.11111111111111, 60.0, 91.85185185185185],

                                      [88.33333333333333, 33.33333333333333, 64.16666666666667, 88.33333333333333],

                                      [91.42857142857143, 55.23809523809524, 80.0, 97.14285714285714],

                                      [92.22222222222223, 40.0, 88.88888888888889, 93.33333333333333],

                                      [94.66666666666667, 65.33333333333333, 88.0, 93.33333333333333]],

                                     [[97.03703703703704, 46.666666666666664, 65.92592592592592, 97.77777777777777],

                                      [95.0, 59.166666666666664, 64.16666666666667, 98.33333333333333],

                                      [97.14285714285714, 63.8095238095238, 96.19047619047619, 99.04761904761905],

                                      [97.77777777777777, 64.44444444444444, 93.33333333333333, 98.88888888888889],

                                      [97.33333333333334, 64.0, 94.66666666666667, 100.0]]]

        # It's ugly but it works. Iterate through the data and display each plot.

        # Order is KNN, SVM-Poly, SVM-rbf, Gaussian. This way I don't have to rerun the

        # classification (slow) to display data (fast)

        for i, accuracies in \

                enumerate(zip(classification\_accuracies, training\_accuracies)):

            classification\_accuracies\_itr = accuracies[0]

            training\_accuracies\_itr = accuracies[1]

            KNN\_accuracies = [x[0] for x in classification\_accuracies\_itr]

            KNN\_train\_accuracies = [x[0] for x in training\_accuracies\_itr]

            SVM\_poly\_accuracies = [x[1] for x in classification\_accuracies\_itr]

            SVM\_poly\_train\_accuracies = [x[1] for x in training\_accuracies\_itr]

            SVM\_rbf\_accuracies = [x[2] for x in classification\_accuracies\_itr]

            SVM\_rbf\_train\_accuracies = [x[2] for x in training\_accuracies\_itr]

            GNB\_accuracies = [x[3] for x in classification\_accuracies\_itr]

            GNB\_train\_accuracies = [x[3] for x in training\_accuracies\_itr]

            plt.plot(test\_sizes, KNN\_accuracies, label="KNN Classification Accuracies")

            plt.plot(test\_sizes, SVM\_poly\_accuracies,label="SVM-Poly Classification Accuracies")

            plt.plot(test\_sizes, SVM\_rbf\_accuracies,label="SVM-rbf Classification Accuracies")

            plt.plot(test\_sizes, GNB\_accuracies,label="Gaussian Classification Accuracies")

            plt.xlabel("% test size")

            plt.ylabel("Average Classification Accuracies")

            if i == 0:

                plt.title("Supervised Classification without Dimensionality Reduction on IRIS")

            elif i == 1:

                plt.title("Supervised Classification with PCA Reduction on IRIS")

            elif i == 2:

                plt.title("Supervised Classification with LDA Reduction on IRIS")

            plt.legend()

            plt.show()

            plt.figure()

            plt.plot(test\_sizes, GNB\_train\_accuracies, label="Gaussian Training Accuracies")

            plt.plot(test\_sizes, KNN\_train\_accuracies, label="KNN Training Accuracies")

            plt.plot(test\_sizes, SVM\_poly\_train\_accuracies, label="SVM-Poly Training Accuracies")

            plt.plot(test\_sizes, SVM\_rbf\_train\_accuracies, label="SVM-rbf Training Accuracies")

            plt.xlabel("% test size")

            plt.ylabel("Average Training Accuracies")

            if i == 0:

                plt.title("Supervised Classification without Dimensionality Reduction on IRIS")

            elif i == 1:

                plt.title("Supervised Classification with PCA Reduction on IRIS")

            elif i == 2:

                plt.title("Supervised Classification with LDA Reduction on IRIS")

            plt.legend()

            plt.show()

def main():

    # Driver for pines and iris data analysis/classification

    filepath = "data/indianR.mat"

    iris\_analysis(display=True)

    pines\_analysis(filepath, display=True)

    iris\_classification(run\_pca=True, run\_lda=False)

    iris\_classification(run\_pca=False, run\_lda=True)

    iris\_classification(run\_pca=False, run\_lda=False)

    pines\_classification(filepath, run\_pca=False, run\_lda=False)

    pines\_classification(filepath, run\_pca=True, run\_lda=False)

    pines\_classification(filepath, run\_pca=False, run\_lda=True)

    plot\_classification(iris=True, pines=True)

if \_\_name\_\_ == "\_\_main\_\_":

    main()