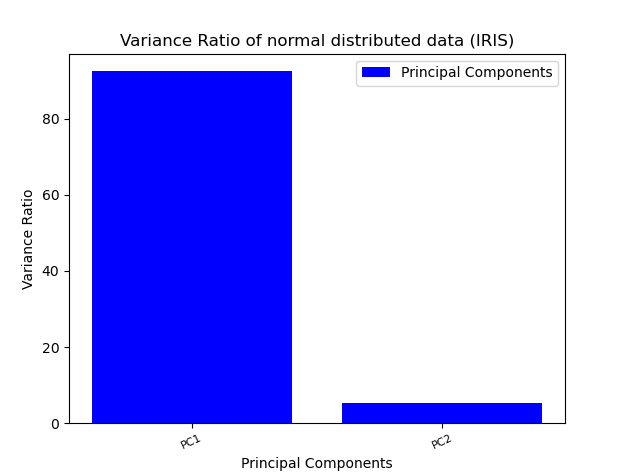
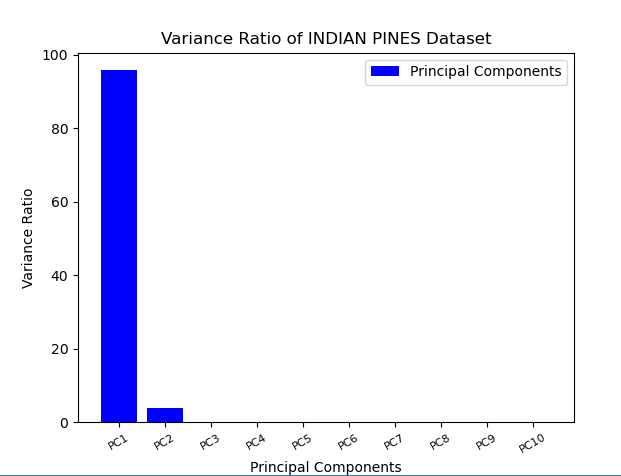
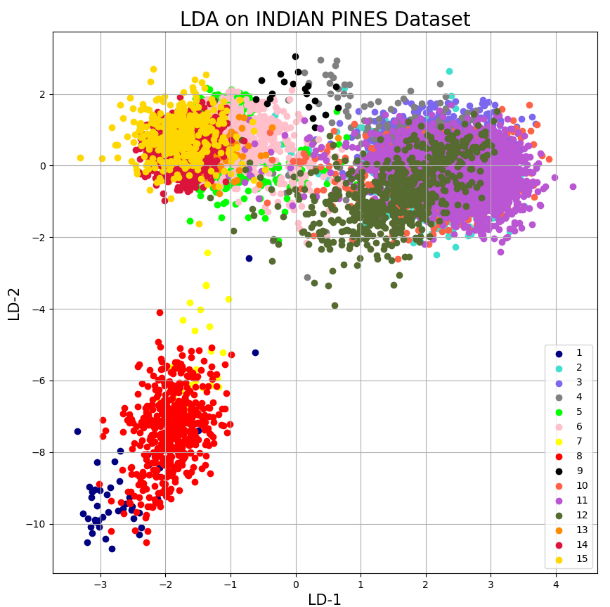
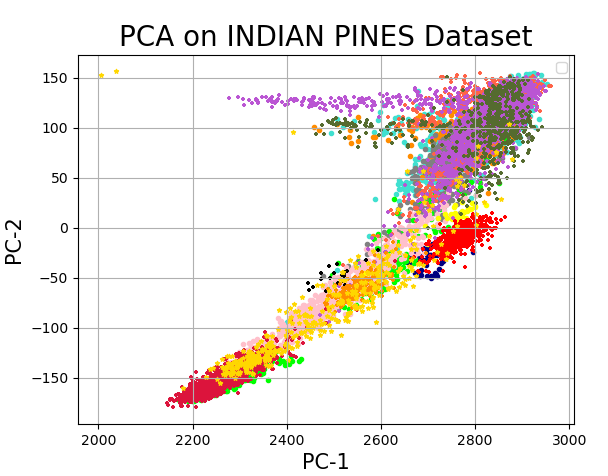
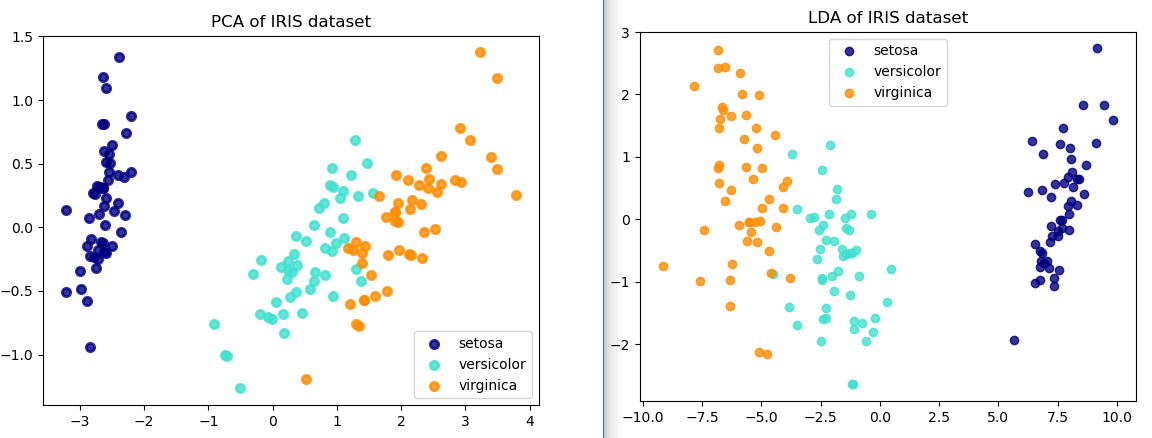
1a) Dimensionality reduction using PCA and LDA on the Iris and Indian Pines dataset

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:

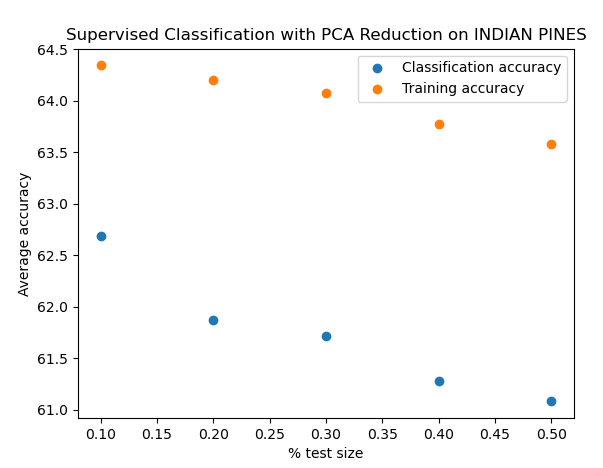
The HSI for the Indian Pines location had 202 layers of pixels for each pixel of ground truth, and only 150x150 pixels of ground truth. To a person, that's basically impossible to visualize the differences between each type of ground pixel. There's too many dimensions to the data to be understandable 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.

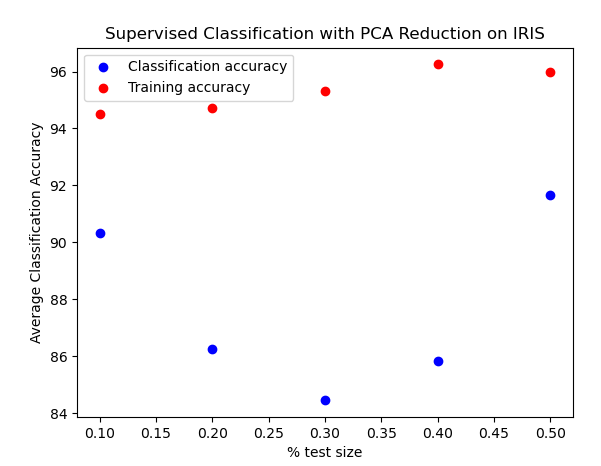
Iris Dataset:

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.

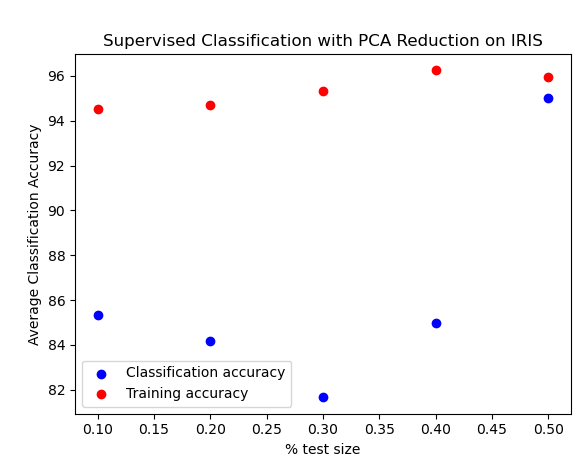
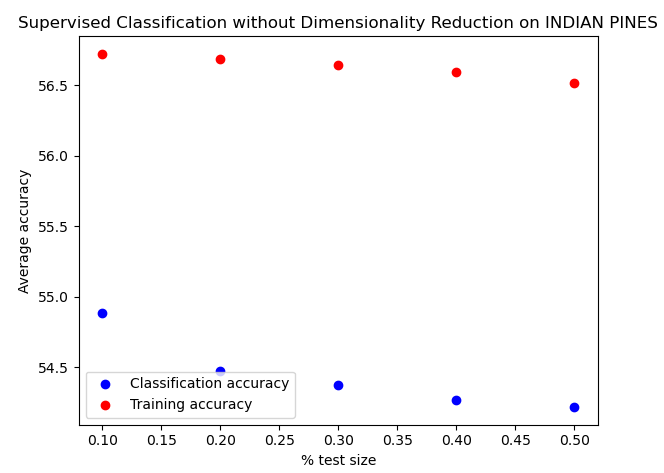
2a) Supervised classification on the Iris and Indian Pines datasets

i) Figures 7 and 8 from top to bottom, plots of average model classification and training accuracy with respect to test size on the Iris dataset

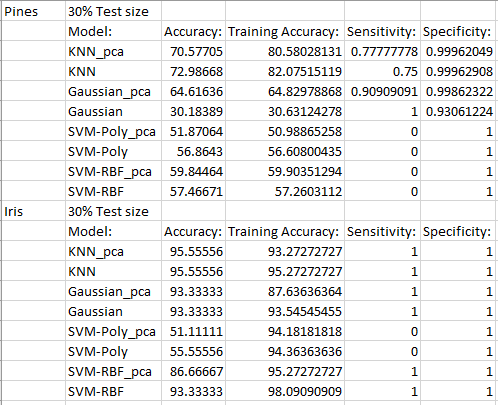




ii) Figures 9 and 10 from top to bottom, plots of average model classification and training accuracy with respect to test size on the Indian Pines dataset



iii) Figure 11, a tabularized class-wise classification accuracy chart for 30% training size over all methods, for both with and without PCA dimensionality reduction for the Indian Pines and Iris datasets



2b) Discuss the analysis of results

Pines Dataset:

With the Pines dataset there was a definite, if slight, improvement of data analysis following the PCA dimensionality reduction compared to running models on the raw data. It increased the classification accuracy from 2-34 percent depending on the model. The effect on the sensitivity and specificity are less noticeable. In some cases it improved, some cases it worsened, and most it remained the same regardless of PCA. The best supervised classification model for the Pines dataset is highlighted in Figure 11, the K-Nearest Neighbors model trained after running PCA dimensionality reduction. In fact, the second best model was the K-Nearest Neighbors without PCA dimensionality reduction. The closest different model was 8 percent behind, the Gaussian NB model with PCA. The support vector machine models did much worse than the KNN and Gaussian, hovering in the 50%'s. This makes sense, because KNN models work better with more complex problems, and HSI with 200+ bands is a very complex problem with 11 different classes. However, the KNN does begin to overfit as seen in the 10% difference between its training accuracy and classification accuracy in Figure 11. A larger dataset or larger training percent would probably increase how well it works. The SVM's probably struggle because there is less data separability, so they aren't able to find distinct lines between classes. The Gaussian model performed the worst, at about 30%. I think the data is not conducive to a Gaussian model because the different classes are not following a statistical curve. There are classes very similar to each other, like different kinds of crops, and classes very different from each other. To me, it seems like this would make Gaussian classification difficult because the features will lean heavily towards multiple classes in a similar way, so they're not all distinct and equally different.

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.

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

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=True):

    # 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)

    test\_sizes = [.1, .2, .3, .4, .5]

    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

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

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

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

            true\_negative = disp.confusion\_matrix[0][0]

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

            # Calculate sensitivity and specificity from confusion matrix

            sensitivity = true\_positive / (true\_positive + false\_negative)

            specificity = true\_negative / (true\_negative + false\_positive)

            # 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

            kfold = StratifiedKFold(n\_splits=10, random\_state=1, shuffle=True)

            cv\_results = cross\_val\_score(model,X\_train,Y\_train,cv=kfold,scoring='accuracy')

            training\_accuracy.append(cv\_results.mean()\*100)

            # Print model information to display

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

            data\_line += " sensitivity: " + str(sensitivity) + " specificity: " + str(specificity) + "\n"

            print(data\_line)

        # Combine all the averages per test size

        plot\_data[test\_size] = [np.mean(training\_accuracy), np.mean(classification\_accuracy)]

    # 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):

    # 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)))

    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 dimensionality reduction")

    else:

        # Display data info

        print("X:", X.shape)

        print("gth:", gth.shape)

        print("Running without dimensionality reduction")

    plot\_data = {}

    test\_sizes = [.5, .4, .3, .2, .1]

    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)

            # Make a confusion matrix

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

            # Grab values for sensitivity/specificty

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

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

            true\_negative = disp.confusion\_matrix[0][0]

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

            sensitivity = true\_positive / (true\_positive + false\_negative)

            specificity = true\_negative / (true\_negative + false\_positive)

            # 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)

            data\_line += " sensitivity: " + str(sensitivity) + " specificity: " + str(specificity) + "\n"

            print(data\_line)

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

        plot\_data[test\_size] = [np.mean(training\_accuracy), np.mean(classification\_accuracy)]

    # 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 = [.5, .4, .3, .2, .1][::-1]

    if pines:

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

        # No PCA

        accuracies = [54.2185865119376, 54.26872770511296, 54.37539632213063, 54.47086801426873, 54.88587731811697][::-1]

        training\_accuracies = [56.51636225266362, 56.59532302814111, 56.64367737990079, 56.68995243757431, 56.72233379135398][::-1]

        plt.scatter(test\_sizes, accuracies, label="Classification accuracy",c="blue")

        plt.scatter(test\_sizes, training\_accuracies, label="Training accuracy",c="red")

        plt.xlabel("% test size")

        plt.ylabel("Average accuracy")

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

        plt.legend()

        plt.show()

        # After PCA

        pca\_accuracies = [61.07914011224199, 61.278240190249704, 61.711318960050725, 61.86682520808561, 62.684260580123635][::-1]

        training\_accuracies = [63.5773401826484, 63.77724930638129, 64.07555887748861, 64.19887039239001, 64.3444139097347][::-1]

        plt.scatter(test\_sizes, pca\_accuracies, label="Classification accuracy")

        plt.scatter(test\_sizes, training\_accuracies, label="Training accuracy")

        plt.xlabel("% test size")

        plt.ylabel("Average accuracy")

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

        plt.legend()

        plt.show()

    if iris:

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

        # No PCA

        training\_accuracies = [95.97527472527473, 96.25, 95.31818181818183, 94.72222222222223, 94.50892857142858][::-1]

        classification\_accuracies = [91.66666666666666, 85.83333333333334, 84.44444444444444, 86.25, 90.33333333333334][::-1]

        plt.scatter(test\_sizes, classification\_accuracies, label="Classification accuracy",c="blue")

        plt.scatter(test\_sizes, training\_accuracies, label="Training accuracy",c="red")

        plt.xlabel("% test size")

        plt.ylabel("Average accuracy")

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

        plt.legend()

        plt.show()

        # After PCA

        training\_accuracies = [95.97527472527473, 96.25, 95.31818181818183, 94.72222222222223, 94.50892857142858][::-1]

        classification\_accuracies =[95.0, 85.0, 81.66666666666666, 84.16666666666666, 85.33333333333333][::-1]

        plt.scatter(test\_sizes, classification\_accuracies, label="Classification accuracy",c="blue")

        plt.scatter(test\_sizes, training\_accuracies, label="Training accuracy",c="red")

        plt.xlabel("% test size")

        plt.ylabel("Average Classification Accuracy")

        plt.title("Supervised Classification with PCA 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)

    iris\_classification(run\_pca=False)

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

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

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

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

    main()