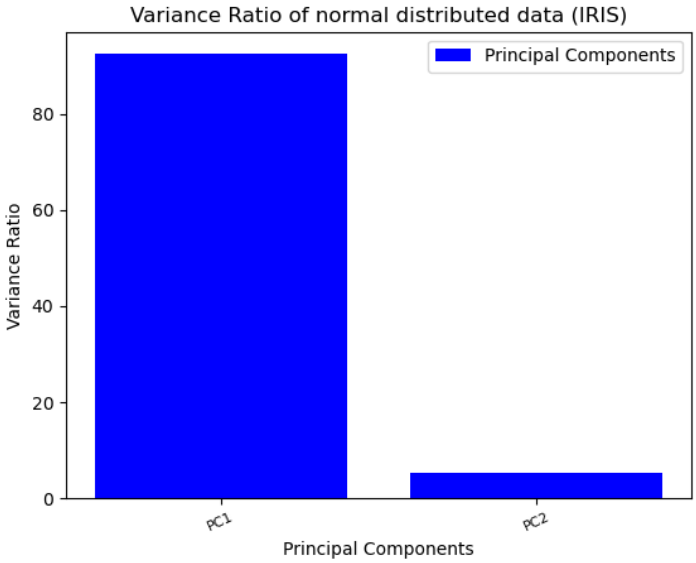
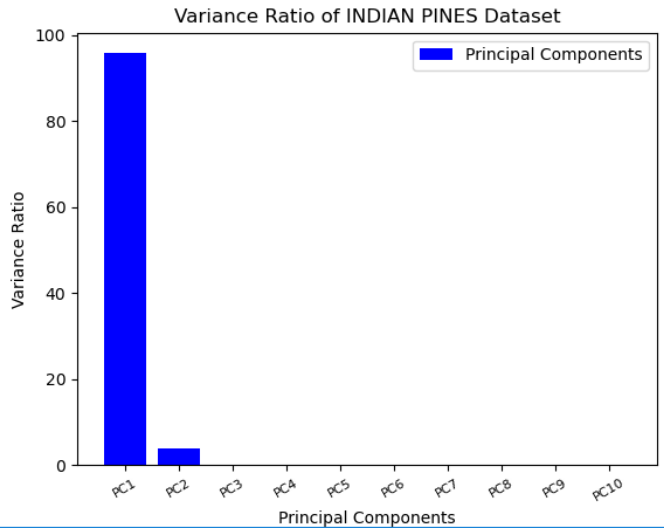
Ian McNichols, CS 488-01, 11/2/2021

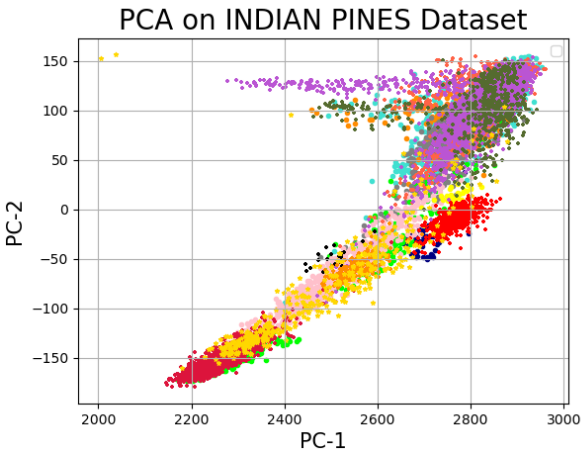
1a) Dimensionality reduction using PCA and LDA on the Iris and Indian Pines dataset

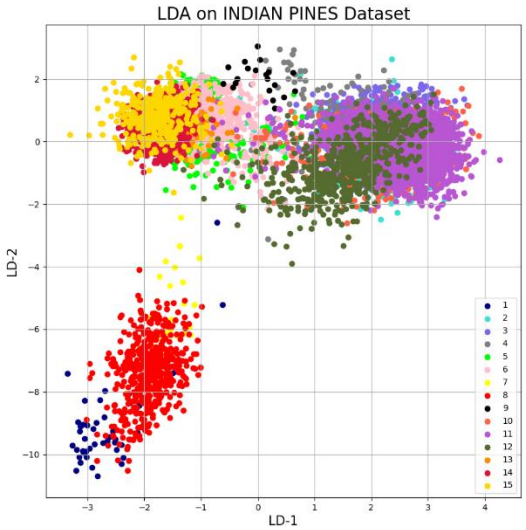
i) Figures 1 and 2 respectively: Plots of variance ratios of the Iris and Indian Pines dataset

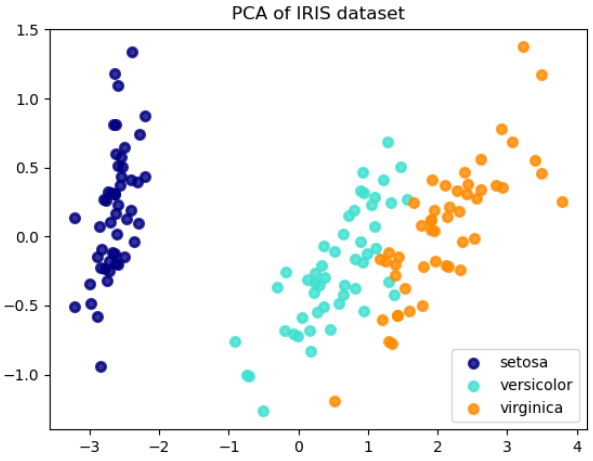


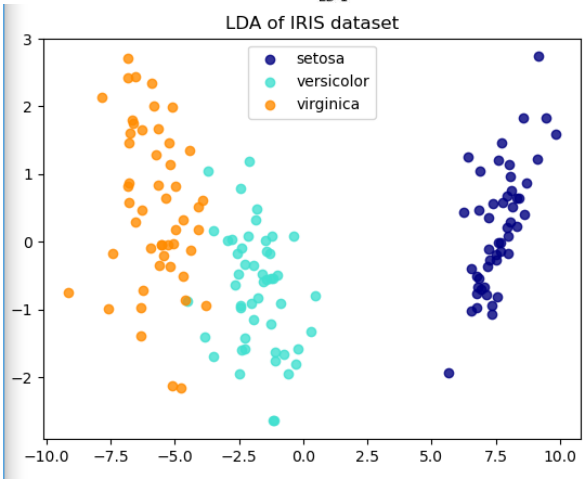


ii) Figures 3, 4, 5, and 6 from top left to bottom right. Plots of 2D PCA and LDA dimensionality reduction on the Indian Pines and then Iris datasets









Ian McNichols, CS 488-01, 11/2/2021

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.

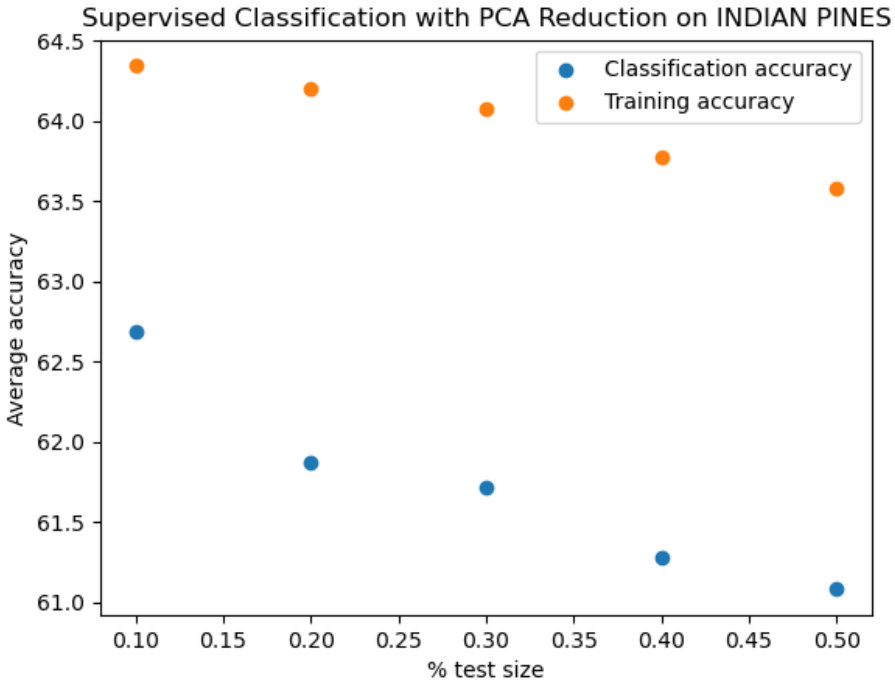
Iris Dataset:

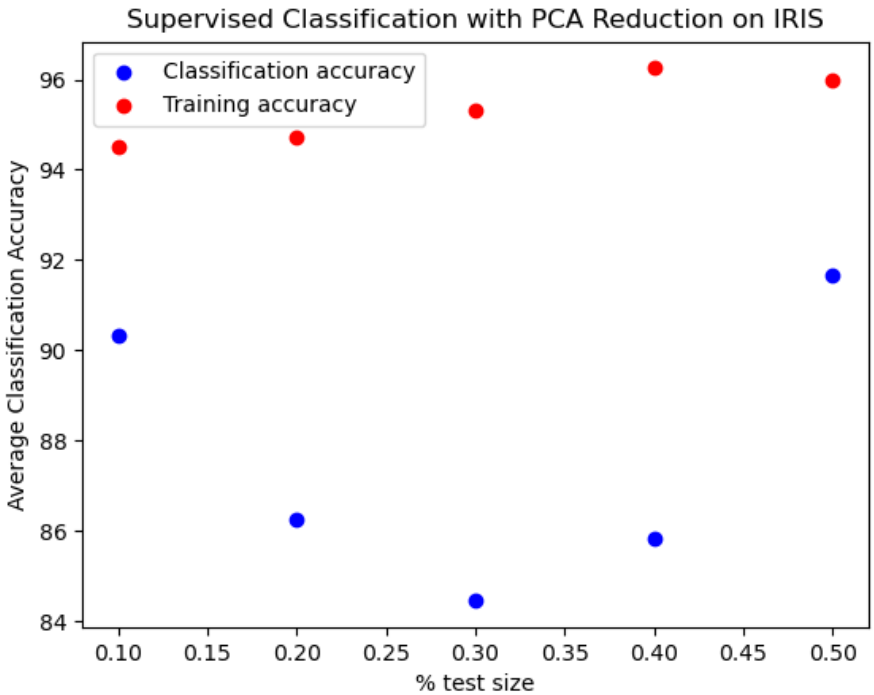
The Iris dataset only has 4 features, so it's almost overkill to run dimensionality on it when you could probably classify the different classes by hand. However, it does enable you to display it in a very concise manner that's easy to understand. On the Iris dataset the clusters are almost entirely 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.

Ian McNichols, CS 488-01, 11/2/2021

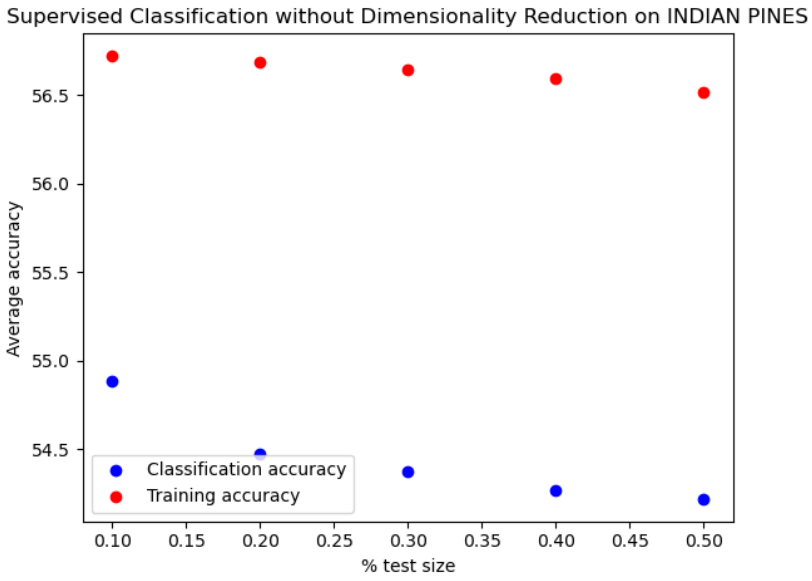
2a) Supervised classification on the Iris and Indian Pines datasets

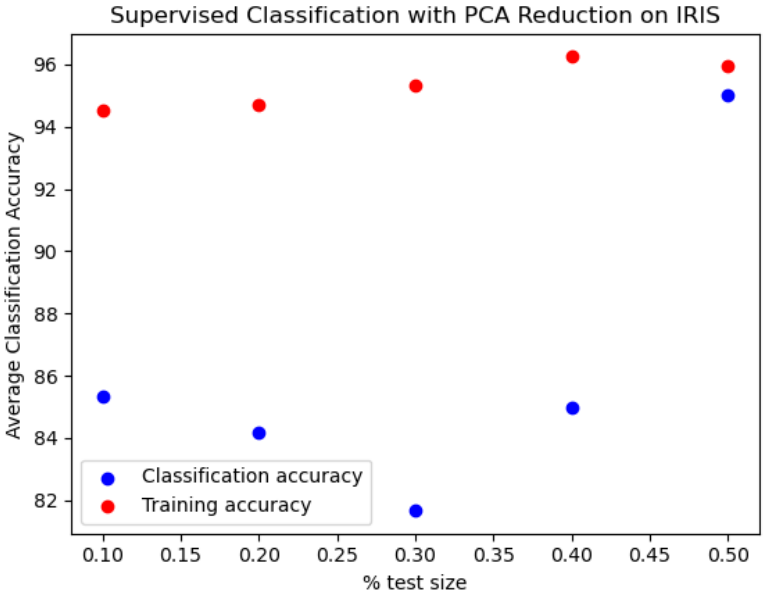
i) Figures 7 and 8 from top to bottom, plots of average model classification and training accuracy with respect to test size on the Iris dataset





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

Pines	30% Test size				
	Model:	Accuracy:	Training Accuracy:	Sensitivity:	Specificity:
	KNN_pca	70.57705	80.58028131	0.77777778	0.99962049
	KNN	72.98668	82.07515119	0.75	0.99962908
	Gaussian_pca	64.61636	64.82978868	0.90909091	0.99862322
	Gaussian	30.18389	30.63124278	1	0.93061224
	SVM-Poly_pca	51.87064	50.98865258	0	1
	SVM-Poly	56.8643	56.60800435	0	1
	SVM-RBF_pca	59.84464	59.90351294	0	1
	SVM-RBF	57.46671	57.2603112	0	1
Iris	30% Test size				
	Model:	Accuracy:	Training Accuracy:	Sensitivity:	Specificity:
	KNN_pca	95.55556	93.27272727	1	1
	KNN	95.55556	95.27272727	1	1
	Gaussian_pca	93.33333	87.63636364	1	1
	Gaussian	93.33333	93.54545455	1	1
	SVM-Poly_pca	51.11111	94.18181818	0	1
	SVM-Poly	55.55556	94.36363636	0	1
	SVM-RBF_pca	86.66667	95.27272727	1	1
	SVM-RBF	93.33333	98.09090909	1	1

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.

Ian McNichols, CS 488-01, 11/2/2021

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_st
ate=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()
```