Final Project

COMP 4910

Maksims Kazijevs

11/20/2020

Summary

In this project, iris.csv will be used as a data frame. Dataframe will consist of 3 different species (Setosa, Virginica, Versicolor) from which only Virginica and Versicolor will be used. Dataframe will have 4 features (Sepal length, Sepal width, Petal length, Petal width) measured in centimeters. We will be working 100x4 data with labels. We will use Python's libraries such as pandas, numpy, sklearn, matplotlib, and time. In the beginning, we will store the desired data in the pandas data frame. From the data frame, we will be able to find the 2 most important principal components to reduce the data by using Principal Component Analysis. We will apply SVM, Neural Network classifier, and K-Means algorithm on the data that we got from principal components. From the results, we will compare the running time of each component and its accuracy in predicting the data.

Definition of Each Component

Principal Component Analysis (PCA) – PCA is a component to reduce the data, it is especially effective when dealing with large datasets in order to reduce it and not lose too much information. PCA finds data with the most variance and gets eigenvalues and eigenvectors. By using these eigenvalues we can get a better picture of the data from principal components.

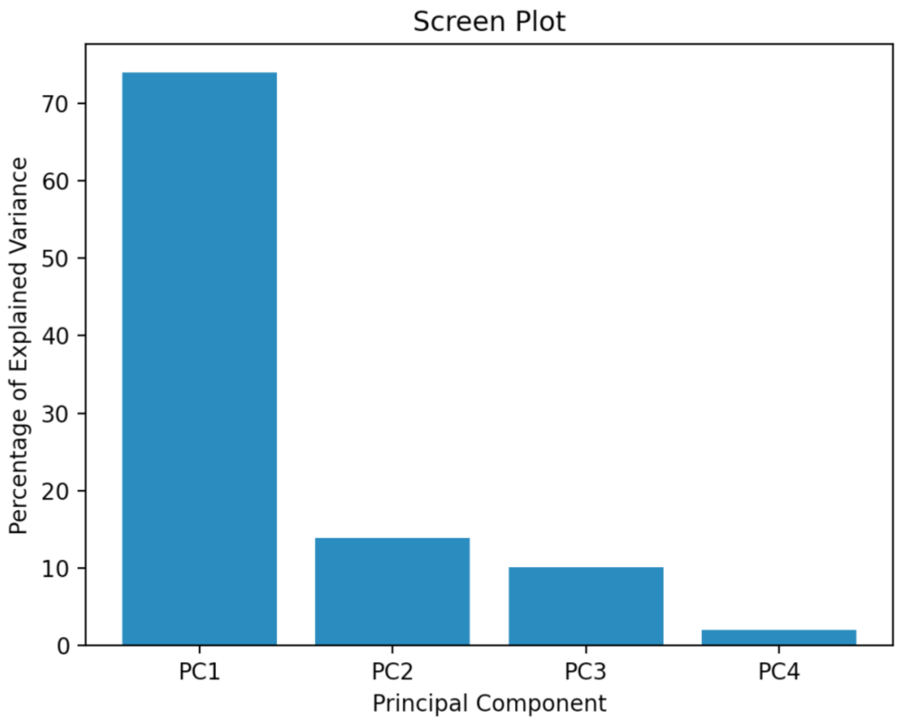
Support Vector Machine (SVM) – SVM is a supervised learning algorithm that is used for classification. The main task for SVM is to find the best line or the plane that separates datasets. Because the data has labels, the algorithm can find the best fitting plane. The algorithm finds the support vectors (closest data to separator) and margins (width between separator and support vectors). The best separator line will have the biggest margin. When the separator will be found we would be able to use data for prediction.

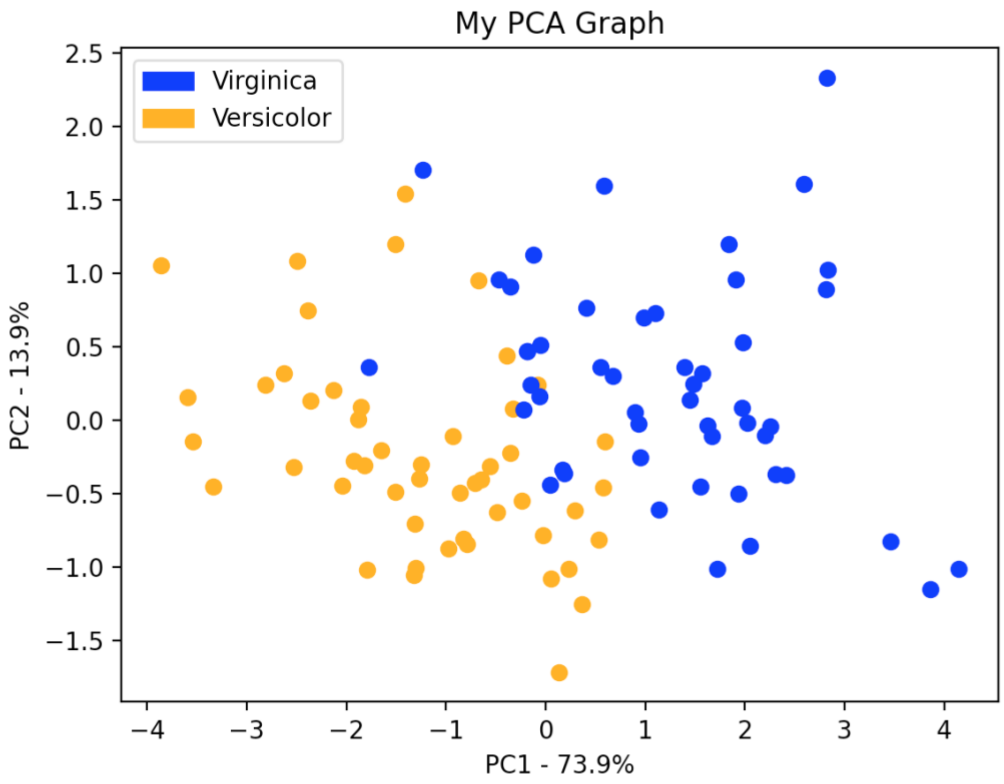
Neural Network – Neural Network is a supervised learning algorithm that is used for prediction. It acts similarly to the human brain. There are different implementations of Neural Networks for different purposes but the common feature in all implementations is that algorithm is learning from the previous results. Behind the algorithm, there are NN layers: input layer (given data), hidden layers (nodes that do computation and find the optimal parameters based on the previous results), output layer (a result that hidden layers produce). Every iteration the data will be changing based on the previous result and the algorithm will put different weights for finding the best output.

K-Means – it is an unsupervised learning algorithm that divides data into different groups (clusters). Each cluster will have the center point (centroid). In the beginning, centroids will be randomly assigned and the distance between data points and centroids will be calculated. The centroids with minimal distance will be assigned to the data points and the cluster will be formed. After, the centroid will update its position with the average position of all data points that belongs to the cluster. The algorithm will be repeated for the desired number of times until it finds the best clusters (when the position of centroid change will be minimized).

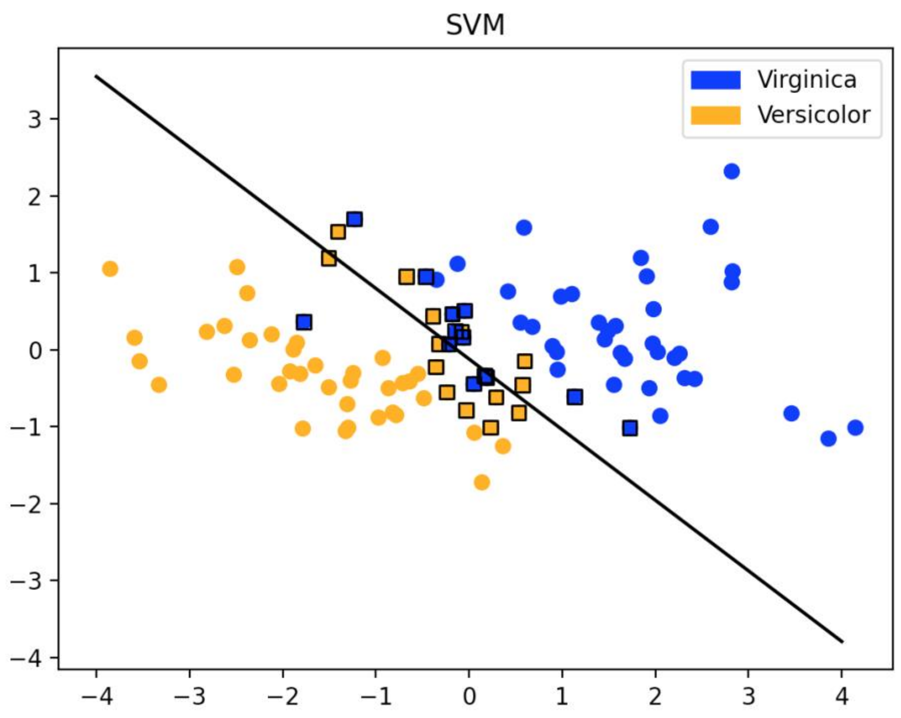
<u>Results</u>

From the PCA we were able to get PCA1 and PCA2 components, we’ve found that PCA1 takes 73.9% of ‘importance’ and PCA2 take 13.9%. And the biggest influence on components produce petal length.

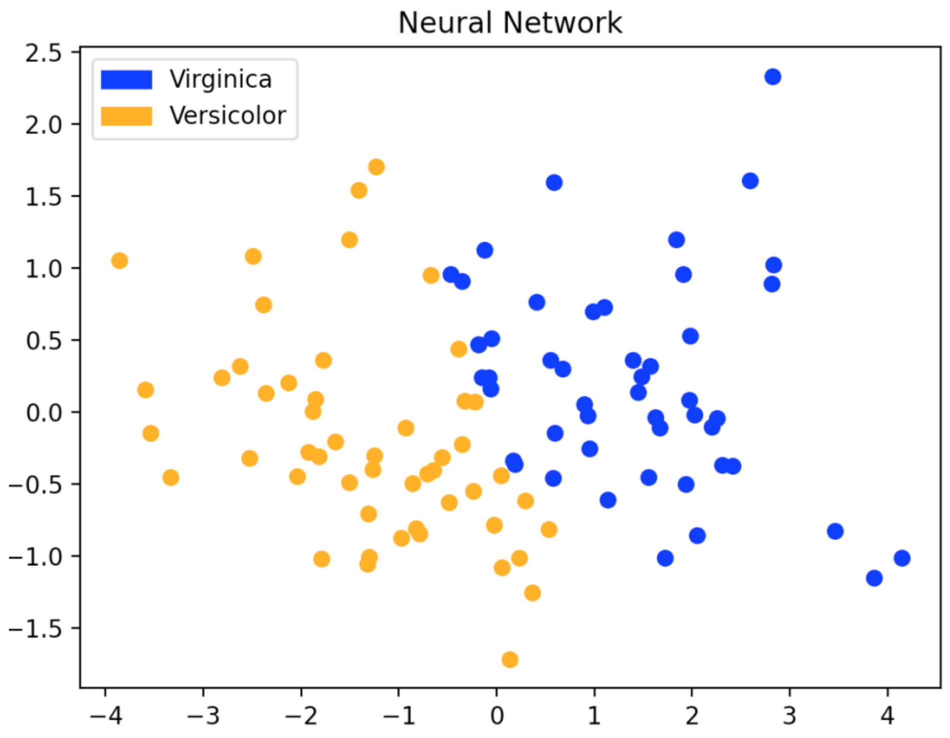




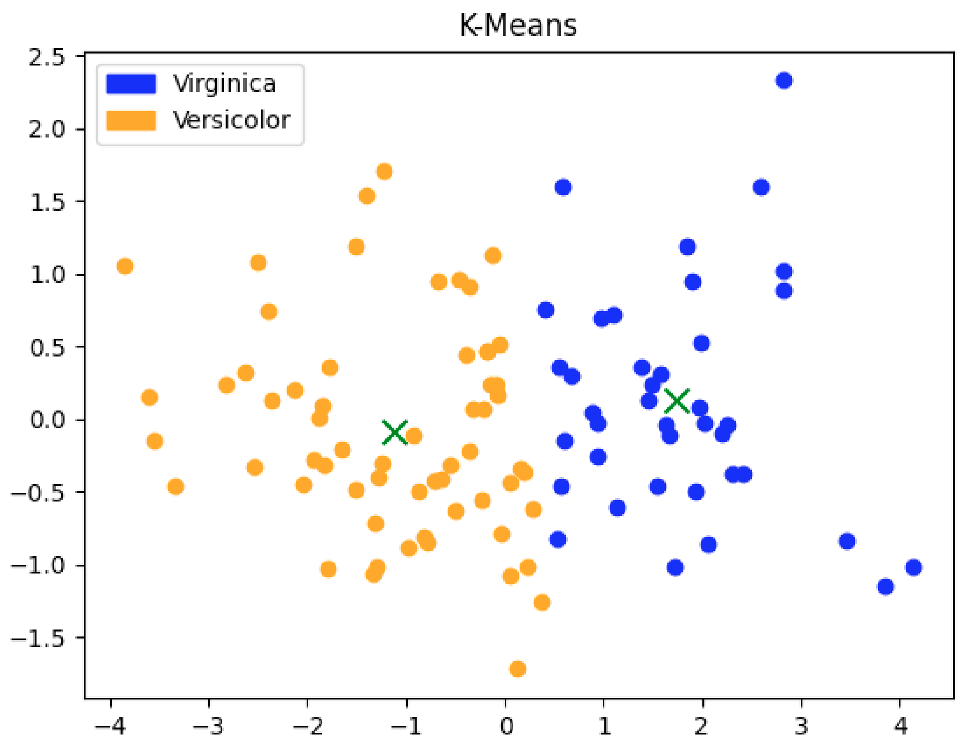
SVM algorithm was able to find classifier, however because the data is too close to each other some data points were classified as another class.



Neural Network algorithm produced a good result in terms of accuracy however it required much more execution time comparing to other algorithms. I have used learning rate with value of 0.0005 and two hidden layers with size of 160 and 140. I’ve put maximum number of iterations to 2000 but 817 were only required. And the final loss value was 0.241



In K-Means, because we knew the number of species that we will use, number of clusters were set to 2, however the result was not satisfactory in terms of predicting precision.



Below will be the table comparing accuracy and execution time for used components

Component	Accuracy	Execution time (seconds)
PCA	N/A	0.532
SVM	89.0%	0.066
Neural Network	93.0%	3.381
K-Means	61.0%	0.483

Note: MacBook Pro Mid 2014 was used for gathering these results (2.8 GHz Intel Core i5, 8GB RAM, Intel Iris 1536MB)

Conclusion

From the 3 approaches used above we could say that K-Means algorithm would be not the best choice to use. Because after applying PCA our data was so close to each other and K-Means couldn’t correctly detect the right clusters with correct data in them. I would say that if we are working with very large data and we don’t have very fast computer it would be better to use SVM, because accuracy percent is high and execution time is very low comparing to Neural Network. If we have very good computer or data set is not very large and we wish to obtain higher accuracy rate Neural Network algorithm will be better choice, but we can see that for 100x4 data set it requires 50 times more time for execution and if data set will be bigger it would require much more time.

Appendix A: Code

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn import preprocessing
from sklearn import svm
from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import time

#Read data from iris.csv and store in the data frame (only virginica and
versicolor)
df = pd.read_csv('iris.csv',

usecols=['sepal_length','sepal_width','petal_length','petal_width','species
'],
                 index_col='species',
                 skiprows=[i for i in range(1,51)])

species = pd.read_csv('iris.csv',
                      usecols=['species'],
                      skiprows=[i for i in range(1,51)]
                      ).T.values.tolist()[0]

#Running time for PCA

begin = time.time()

#Applying PCA
scaled_data = preprocessing.scale(df)
pca = PCA()
pca.fit(scaled_data)
pca_data = pca.transform(scaled_data)

#Finding PCA components
per_var = np.round(pca.explained_variance_ratio_ * 100, decimals=1)
labels = ['PC' + str(x) for x in range(1, len(per_var) + 1)]

plt.bar(x=range(1, len(per_var) + 1), height=per_var, tick_label=labels)
plt.ylabel('Percentage of Explained Variance')
plt.xlabel('Principal Component')
plt.title('Screen Plot')
plt.show()

pca_df = pd.DataFrame(pca_data, index=species, columns=labels)

#Plotting PCA data
plt.figure(2)
plt.title('My PCA Graph')
plt.xlabel('PC1 - {0}%'.format(per_var[0]))
plt.ylabel('PC2 - {0}%'.format(per_var[1]))

i = 0
flower = ["versicolor","virginica"]
color = ["orange","blue"]
for sample in pca_df.index:
    if sample == flower[0]:
        curColor = color[0]
    if sample == flower[1]:
        curColor = color[1]

    plt.scatter(pca_df.PC1[i], pca_df.PC2[i], c = curColor)
    i = i+1

orange_patch = mpatches.Patch(color='orange', label='Versicolor')
blue_patch = mpatches.Patch(color='blue', label='Virginica')
plt.legend(handles=[blue_patch, orange_patch ])

# Determine which grades had the biggest influence on PC1
loading_scores = pd.Series(pca.components_[0],
index=['sepal_length','sepal_width','petal_length','petal_width'])

## Sort the scores
sorted_loading_scores = loading_scores.abs().sort_values(ascending=False)

# Show the names of the grades
top_4_features = sorted_loading_scores[0:10].index.values

print("---PCA---")
print("Execution time for PCA:", time.time() - begin,"seconds")
print("Most important principal components:\n"
      "PC1:", per_var[0],
      "PC2:", per_var[1])
print("Biggest influence features:")
print(loading_scores[top_4_features])
plt.show()
#SVM

#Prepare data for training
pc1 = pca_df['PC1'].to_numpy()
pc2 = pca_df['PC2'].to_numpy()

#Running time for SVM

begin = time.time()

#Training
training_X = np.vstack((pc1, pc2)).T
# preparing the labeling
training_Y = np.array(species)

clf = svm.SVC(kernel='linear', C=1.0)
clf.fit(training_X, training_Y)

# weights
w = clf.coef_[0]

# offset
a = -w[0] / w[1]

XX = np.linspace(-4, 4)
yy = a * XX - clf.intercept_[0] / w[1]

#Plotting data
plt.figure(3)
plt.plot(XX, yy, 'k-')
plt.scatter(pc1[0:50], pc2[0:50], color='orange')
plt.scatter(pc1[50:100], pc2[50:100], color='blue')
plt.scatter(clf.support_vectors_[:,0], clf.support_vectors_[:,1],
marker='s',edgecolors='black', facecolors='none')

orange_patch = mpatches.Patch(color='orange', label='Versicolor')
blue_patch = mpatches.Patch(color='blue', label='Virginica')
plt.legend(handles=[blue_patch, orange_patch ])
plt.title('SVM')

#Testing accuracy
index = 0
correct = 0
incorrect = 0
for item in training_X:
    #print(clf.predict([[item[0],item[1]]]))
    #print(training_y[index])
    if clf.predict([[item[0],item[1]]]) == training_Y[index]:
        correct += 1
    else:
        incorrect += 1
    index += 1


print("\n---SVM---")
print("Execution time for SVM:", time.time() - begin, "seconds")
print("Accuracy of prediction of SVM:", (correct/(correct+incorrect))*100,
"%")
plt.show()

#Neural Network

#Running time for Neural Network

begin = time.time()

#Training
nn = MLPClassifier(solver='sgd',
learning_rate='constant',learning_rate_init=0.0005,
hidden_layer_sizes=(160,140), max_iter =2000, random_state=1)
nn.fit(training_X, training_Y )

# Plotting Neural Network results
plt.figure(4)
for item in training_X:
    ans = nn.predict([[item[0],item[1]]])
    if ans=='versicolor':
        plt.scatter(item[0], item[1], color='orange')
    else:
        plt.scatter(item[0], item[1], color='blue')
plt.title('Neural Network')
orange_patch = mpatches.Patch(color='orange', label='Versicolor')
blue_patch = mpatches.Patch(color='blue', label='Virginica')
plt.legend(handles=[blue_patch, orange_patch ])
plt.show()


#Comparing the predictions against the actual observations
yp = nn.predict(training_X)

#check how many of them are predicted well
count  = 0;
for i in range(len(training_Y)):
    if yp[i] == training_Y[i]:
        count +=1

accuracy = count/len(training_Y)*100

#Printing the accuracy
print("\n---Neural Network---")
print("Execution time for Neural Network:", time.time() - begin, "seconds")
print('Accuracy of MLPClassifier(in percentage) :', accuracy)
print('Final Loss Value :', nn.loss_)
print('Final Iterations :', nn.n_iter_)

#K-Means

#Running time for K-Means

begin = time.time()

#Need only 2 clusters
kmeans = KMeans(n_clusters=2, random_state=0, max_iter=200)
kmeans.fit(training_X)


#Plotting K-Means data
plt.figure(5)
i = 0
for item in training_X:
    if kmeans.labels_[i]==0:
        plt.scatter(item[0], item[1], color='orange')
    else:
        plt.scatter(item[0], item[1], color='blue')
    i += 1
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s=100, marker='x', c='green')
orange_patch = mpatches.Patch(color='orange', label='Versicolor')
blue_patch = mpatches.Patch(color='blue', label='Virginica')
plt.legend(handles=[blue_patch, orange_patch ])
plt.title('K-Means')
plt.show()

#Testing accuracy
index = 0
correct = 0
incorrect = 0
for item in training_X:
    if kmeans.predict([[item[0],item[1]]]) == 0:
        correct += 1
    else:
        incorrect += 1
    index += 1

# results
print("\n---K-Means---")
print("Execution time for PCA:", time.time() - begin, "seconds")
print("K-Means score:",kmeans.score(training_X))
print("Accuracy of prediction of K-Means:",
(correct/(correct+incorrect))*100, "%")
```