

**American University of Sharjah**

**College of Engineering**

**Department of Computer Science & Engineering**

**Spring 2021**

**CMP 466 – Machine Learning & Data Mining**

**Assignment 5**

**Farooq Mirza 80205**

**Contents**

[**Question 1 3**](#_Toc72610857)

[**Question 2 3**](#_Toc72610858)

[**Question 3 6**](#_Toc72610859)

[**Question 4 7**](#_Toc72610860)

[**Question 5 9**](#_Toc72610861)

[**Question 6 12**](#_Toc72610862)

[**Question 7 14**](#_Toc72610863)

# Question 1

# I performed MinMaxScaler on the dataset, the Wisconsin breast cancer dataset to make all the values scaled to be between the range of 0 to 1 by specifying the feature\_range parameter to be equal to (0,1).

#-----------------------

# Reading the data

#-----------------------

data = pd.read\_csv('Breast Cancer Wisconsin Data Set')

data = data.drop(columns=['id', 'Unnamed: 32'])

data = data.replace(['M','B'],[0, 1])

#-----------------------

# Scaling the data

#-----------------------

Min\_max = preprocessing.MinMaxScaler(feature\_range = (0,1))

cols = data.columns

# print(data)

data[cols] = Min\_max.fit\_transform(data[cols])

# print(data)

**Question 2**

From the assignments that I have previously submitted, I have obtained the highest testing accuracies for the Support Vector Machine Classifier when the kernel is Linear and the Gaussian Naïve Bayes Classifier with default parameters. For this assignment I have decided to use random splitting as a change since I performed k-fold cross validation for these models in earlier assignments. For both the models, the training and testing accuracy were calculated separately to obtain the time required to fit each of them. After this, I used the precision\_recall\_fscore\_support function from sklearn to get the Precision, Recall and F-Score for both the Malignant and Benign classes.

#-----------------------

# Splitting the data

#-----------------------

X = data.drop(columns=['diagnosis']).astype(float).to\_numpy()

y = data['diagnosis'].astype(int).to\_numpy()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 42)

#-----------------------

# 2 Best Classifiers for Dataset

#-----------------------

# SVC Kernel=Linear

print("\nSVC Kernel Linear")

svc\_lin\_clf = svm.SVC(kernel='linear')

start = time.time()

svc\_lin\_clf.fit(X\_train, y\_train)

end = time.time()

print("\nTraining Time = ", end-start)

print("Training accuracy =", svc\_lin\_clf.score(X\_train, y\_train))

start = time.time()

svc\_lin\_clf.fit(X\_test, y\_test)

end = time.time()

print("\nTesting Time = ", end-start)

print("Testing accuracy =", svc\_lin\_clf.score(X\_test, y\_test))

pre\_rec\_fsc = precision\_recall\_fscore\_support(y\_test, svc\_lin\_clf.predict(X\_test))

print("\nMalignant Precision =", pre\_rec\_fsc[0][0], ", Benign Precision =", pre\_rec\_fsc[0][1])

print("\nMalignant Recall =", pre\_rec\_fsc[1][0], ", Benign Recall =", pre\_rec\_fsc[1][1])

print("\nMalignant F-Score =", pre\_rec\_fsc[2][0], ", Benign F-Score =", pre\_rec\_fsc[2][1])

# Guassian Naive Bayes

print("\nGuassian Naive Bayes")

gnb\_clf = GaussianNB()

start = time.time()

gnb\_clf.fit(X\_train, y\_train)

end = time.time()

print("\nTraining Time = ", end-start)

print("Training accuracy =", gnb\_clf.score(X\_train, y\_train))

start = time.time()

gnb\_clf.fit(X\_test, y\_test)

end = time.time()

print("\nTesting Time = ", end-start)

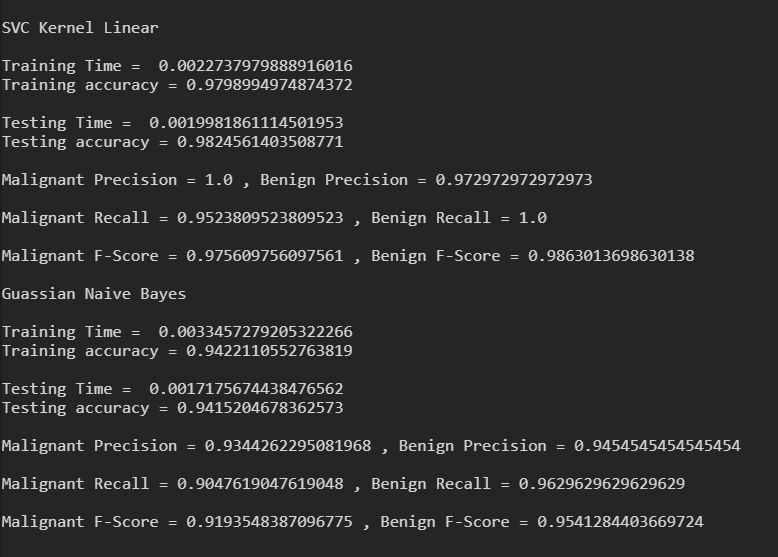
print("Testing accuracy =", gnb\_clf.score(X\_test, y\_test))

pre\_rec\_fsc = precision\_recall\_fscore\_support(y\_test, gnb\_clf.predict(X\_test))

print("\nMalignant Precision =", pre\_rec\_fsc[0][0], ", Benign Precision =", pre\_rec\_fsc[0][1])

print("\nMalignant Recall =", pre\_rec\_fsc[1][0], ", Benign Recall =", pre\_rec\_fsc[1][1])

print("\nMalignant F-Score =", pre\_rec\_fsc[2][0], ", Benign F-Score =", pre\_rec\_fsc[2][1])



# Question 3

# I opted to use SelectKBest for the feature selection. I opted for the hyperparameter k = 10, after trying a few values (2, 4, 6, 10, 15, 20, 25 and 100) that gave me the best results through trial and error.

#-----------------------

# Feature Selection

#-----------------------

print("\n\nFeature Selection")

X\_best = SelectKBest(chi2, k=10).fit\_transform(X, y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_best, y, test\_size = 0.3, random\_state = 42)

**Question 4**

After obtaining the X\_train, X\_test, y\_train, y\_test for the k best features, I performed the same steps as in Question 2 to obtain the Training time and accuracy, Testing time and accuracy, Precision, Recall and F-Score for both the Malignant and Benign classes.

# SVC Kernel=Linear

print("\nSVC Kernel Linear")

svc\_lin\_clf = svm.SVC(kernel='linear')

start = time.time()

svc\_lin\_clf.fit(X\_train, y\_train)

end = time.time()

print("\nTraining Time = ", end-start)

print("Training accuracy =", svc\_lin\_clf.score(X\_train, y\_train))

start = time.time()

svc\_lin\_clf.fit(X\_test, y\_test)

end = time.time()

print("\nTesting Time = ", end-start)

print("Testing accuracy =", svc\_lin\_clf.score(X\_test, y\_test))

pre\_rec\_fsc = precision\_recall\_fscore\_support(y\_test, svc\_lin\_clf.predict(X\_test))

print("\nMalignant Precision =", pre\_rec\_fsc[0][0], ", Benign Precision =", pre\_rec\_fsc[0][1])

print("\nMalignant Recall =", pre\_rec\_fsc[1][0], ", Benign Recall =", pre\_rec\_fsc[1][1])

print("\nMalignant F-Score =", pre\_rec\_fsc[2][0], ", Benign F-Score =", pre\_rec\_fsc[2][1])

# Guassian Naive Bayes

print("\nGuassian Naive Bayes")

gnb\_clf = GaussianNB()

start = time.time()

gnb\_clf.fit(X\_train, y\_train)

end = time.time()

print("\nTraining Time = ", end-start)

print("Training accuracy =", gnb\_clf.score(X\_train, y\_train))

start = time.time()

gnb\_clf.fit(X\_test, y\_test)

end = time.time()

print("\nTesting Time = ", end-start)

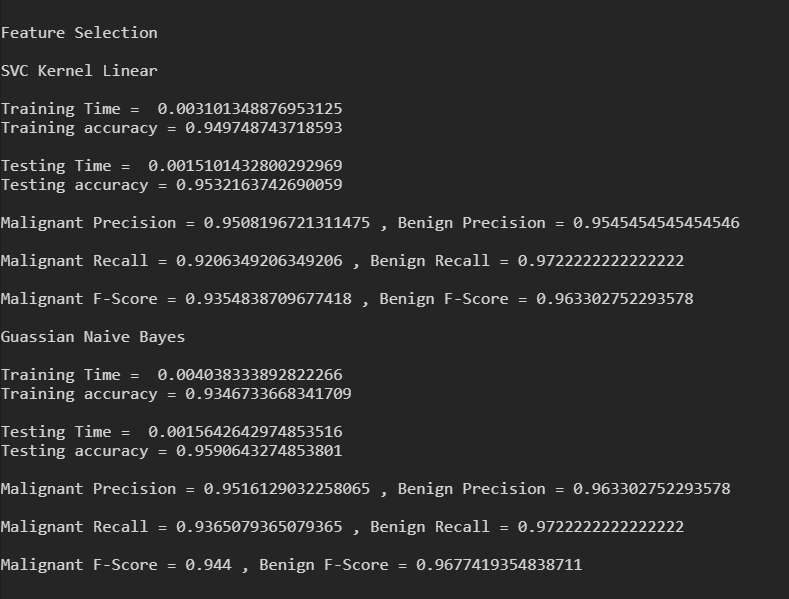
print("Testing accuracy =", gnb\_clf.score(X\_test, y\_test))

pre\_rec\_fsc = precision\_recall\_fscore\_support(y\_test, gnb\_clf.predict(X\_test))

print("\nMalignant Precision =", pre\_rec\_fsc[0][0], ", Benign Precision =", pre\_rec\_fsc[0][1])

print("\nMalignant Recall =", pre\_rec\_fsc[1][0], ", Benign Recall =", pre\_rec\_fsc[1][1])

print("\nMalignant F-Score =", pre\_rec\_fsc[2][0], ", Benign F-Score =", pre\_rec\_fsc[2][1])



# Question 5

# I applied PCA on the dataset, to obtain the first few principal components that amount to more than 70% of the variance.

#-----------------------

# Principal Component Analysis

#-----------------------

print("\n\nPrincipal Component Analysis (n=30)")

pca = PCA(n\_components=30)

pca.fit(X)

# exp\_var\_pca = np.arange(0, len(pca.explained\_variance\_ratio\_), step = 1)

cum\_sum\_eigenvalues = np.cumsum(pca.explained\_variance\_ratio\_)

print("\nSingular Values:\n", pca.singular\_values\_)

print("\nExplained Variance Ratio:\n", pca.explained\_variance\_ratio\_)

print("\nCumulative Variance:\n", cum\_sum\_eigenvalues)

exp\_var\_pca = pca.explained\_variance\_ratio\_

cum\_sum\_eigenvalues = np.cumsum(exp\_var\_pca)

plt.figure(1)

plt.xticks(np.arange(0, 30, 1), fontsize=8)

plt.bar(range(0,len(exp\_var\_pca)), exp\_var\_pca, alpha=0.5, align='center', label='Individual explained variance')

plt.step(range(0,len(cum\_sum\_eigenvalues)), cum\_sum\_eigenvalues, where='mid',label='Cumulative explained variance')

plt.ylabel('Explained variance ratio')

plt.xlabel('Principal component index')

plt.axhline(y=0.7, color='r', label="Variance = 0.7")

plt.legend()

# 

# 

# Looking at this plot we can see that the cumulative variance goes above 70% at x-axis value of 1. This is also clear from looking at the cumulative sum at index = 1 being 0.70381179. Therefore, we can say that the first 2 components explain over 70% of the variance. Therefore, I took the hyperparameter n\_components = 2 in the proceeding part for creating the X\_train, X\_test, y\_train, y\_test splits.

print("\n\nPrincipal Component Analysis (n=2)")

pca = PCA(n\_components=2)

pca.fit(X)

X\_pca = pca.fit\_transform(X)

print("\nExplained Variance Ratio:\n", pca.explained\_variance\_ratio\_)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_pca, y,  test\_size = 0.3, random\_state = 42)

**Question 6**

I performed the same steps as in Question 2 on the newly generated splits using PCA to obtain the Training time and accuracy, Testing time and accuracy, Precision, Recall and F-Score for both the Malignant and Benign classes.

# SVC Kernel=Linear

print("\nSVC Kernel Linear")

svc\_lin\_clf = svm.SVC(kernel='linear')

start = time.time()

svc\_lin\_clf.fit(X\_train, y\_train)

end = time.time()

print("\nTraining Time = ", end-start)

print("Training accuracy =", svc\_lin\_clf.score(X\_train, y\_train))

start = time.time()

svc\_lin\_clf.fit(X\_test, y\_test)

end = time.time()

print("\nTesting Time = ", end-start)

print("Testing accuracy =", svc\_lin\_clf.score(X\_test, y\_test))

pre\_rec\_fsc = precision\_recall\_fscore\_support(y\_test, svc\_lin\_clf.predict(X\_test))

print("\nMalignant Precision =", pre\_rec\_fsc[0][0], ", Benign Precision =", pre\_rec\_fsc[0][1])

print("\nMalignant Recall =", pre\_rec\_fsc[1][0], ", Benign Recall =", pre\_rec\_fsc[1][1])

print("\nMalignant F-Score =", pre\_rec\_fsc[2][0], ", Benign F-Score =", pre\_rec\_fsc[2][1])

# Guassian Naive Bayes

print("\nGuassian Naive Bayes")

gnb\_clf = GaussianNB()

start = time.time()

gnb\_clf.fit(X\_train, y\_train)

end = time.time()

print("\nTraining Time = ", end-start)

print("Training accuracy =", gnb\_clf.score(X\_train, y\_train))

start = time.time()

gnb\_clf.fit(X\_test, y\_test)

end = time.time()

print("\nTesting Time = ", end-start)

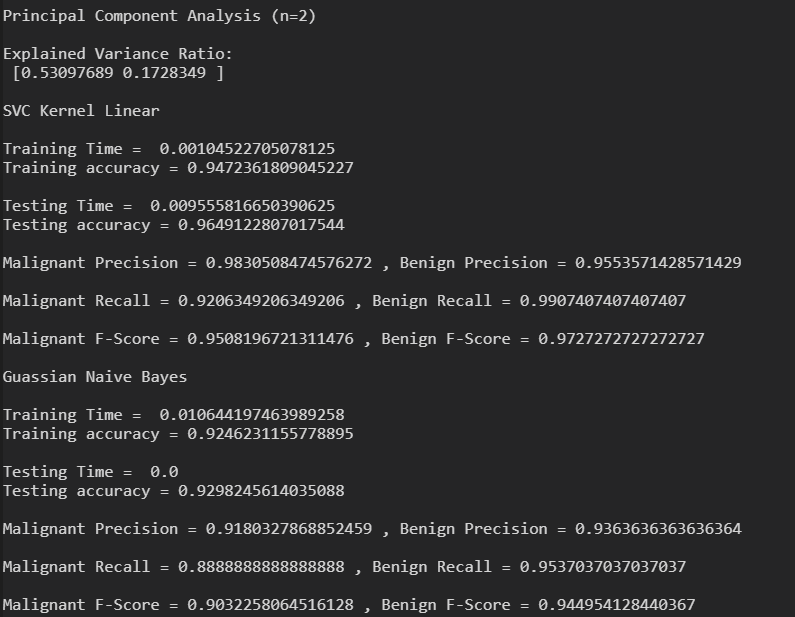
print("Testing accuracy =", gnb\_clf.score(X\_test, y\_test))

pre\_rec\_fsc = precision\_recall\_fscore\_support(y\_test, gnb\_clf.predict(X\_test))

print("\nMalignant Precision =", pre\_rec\_fsc[0][0], ", Benign Precision =", pre\_rec\_fsc[0][1])

print("\nMalignant Recall =", pre\_rec\_fsc[1][0], ", Benign Recall =", pre\_rec\_fsc[1][1])

print("\nMalignant F-Score =", pre\_rec\_fsc[2][0], ", Benign F-Score =", pre\_rec\_fsc[2][1])



# Question 7

We can notice that in all scenarios and metrics, the SVC Linear Kernel Classifier relatively outperforms the Gaussian Naïve Bayes Classifier except in the feature selection where the Gaussian Naïve Bayes Classifier has a small lead in all the performance metrics. When I obtain the performance metrics for the classifiers for the first time, the SVC Linear Kernel is quicker in training as well as testing and out-performs the Gaussian Naïve Bayes Classifier in the Precision, Recall and F-Score for both Benign and Malignant classes. This is also the case in the Principal Component Analysis with 2 principal components. It is however to be noted that the Naïve bayes classifier performed better in feature selection, although the relative difference is very small.

Overall, we can conclude that the SVC having Linear Kernel performs slightly better than the Naïve Bayes approach due to it performing better in normal splits as well as in Principal Component Analysis while only taking two thirds the training and testing times to fit the data.