

**American University of Sharjah**

**College of Engineering**

**Department of Computer Science & Engineering**

**Spring 2021**

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

**Assignment 4**

**Farooq Mirza 80205**

**Table of Contents**

[**Question 1**](#_jdphag8j5iqz)

[**Question 2**](#_66pay6cny5t9)

[**Question 3**](#_d0wuwkyrrd2q)

[**Question 4**](#_8qnemewpfe00)

[**Question 5**](#_9mdtp13u9tmv)

# **Question 1:**

I performed K- Nearest Neighbours Classifier on the our dataset, the Wisconsin breast cancer dataset, using 5-folds. This is due to the fact that our dataset performed slightly better in 5 folds than in 10 folds as discussed in our previous assignment. I tweaked the nearest neighbour hyper parameter between 1 and 15, cross validated the knn classifier with the scoring hyper parameter set for accuracy, precision, recall and f-score metrics. I then proceeded to store the average value for all folds in each iteration into arrays for both training and testing to plot them for easier understanding of all the performance metrics.

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

# KNN Classifier

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

scoring = ['accuracy', 'precision\_macro', 'recall\_macro', 'f1\_macro']

tr\_acc = []

te\_acc = []

tr\_pre = []

te\_pre = []

tr\_rec = []

te\_rec = []

tr\_f1s = []

te\_f1s = []

neighbors = []

print("KNN Classifier:")

for i in range(1,15):

knn\_clf = KNeighborsClassifier(n\_neighbors=i)

knn\_clf.fit(X, y)

cv\_results = cross\_validate(knn\_clf, X, y, cv=5, scoring=scoring, return\_train\_score=True)

tr\_acc.append(np.average(cv\_results['train\_accuracy']))

te\_acc.append(np.average(cv\_results['test\_accuracy']))

tr\_pre.append(np.average(cv\_results['train\_precision\_macro']))

te\_pre.append(np.average(cv\_results['test\_precision\_macro']))

tr\_rec.append(np.average(cv\_results['train\_recall\_macro']))

te\_rec.append(np.average(cv\_results['test\_recall\_macro']))

tr\_f1s.append(np.average(cv\_results['train\_f1\_macro']))

te\_f1s.append(np.average(cv\_results['test\_f1\_macro']))

neighbors.append(i)

plt.figure(1)

plt.plot(neighbors, tr\_acc, label='Training')

plt.plot(neighbors, te\_acc, label='Testing')

plt.xlabel("Neighbours")

plt.ylabel("Accuracy")

plt.legend()

plt.figure(2)

plt.plot(neighbors, tr\_pre, label='Training')

plt.plot(neighbors, te\_pre, label='Testing')

plt.xlabel("Neighbours")

plt.ylabel("Precision")

plt.legend()

plt.figure(3)

plt.plot(neighbors, tr\_rec, label='Training')

plt.plot(neighbors, te\_rec, label='Testing')

plt.xlabel("Neighbours")

plt.ylabel("Recall")

plt.legend()

plt.figure(4)

plt.plot(neighbors, tr\_f1s, label='Training')

plt.plot(neighbors, te\_f1s, label='Testing')

plt.xlabel("Neighbours")

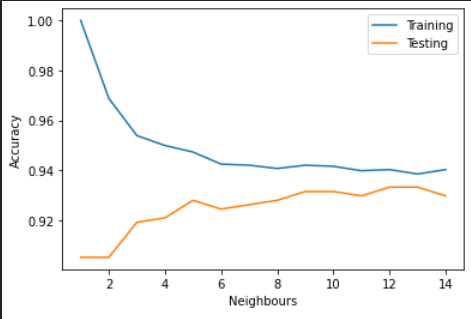
plt.ylabel("F Score")

plt.legend()

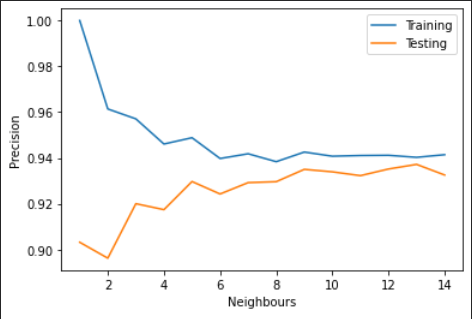
## **Question 2:**

As mentioned in the previous part, I calculated and constructed plots for the training accuracy, testing accuracy, recall, precision, F-score, followed by the confusion matrix.

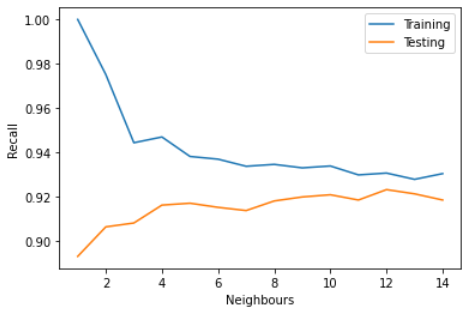
**Average Accuracy vs Number of Neighbours**

****

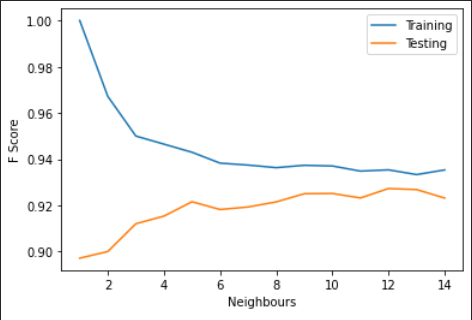
**Average Precision vs Number of Neighbours**

****

**Average Recall vs Number of Neighbours**

****

**Average F-Score vs Number of Neighbours**

****

The plots suggest that the performance metrics are optimal for the number of neighbours equal to 13, as this is the region in the plot where the testing and training performance metrics come closest to each other. I then made a confusion matrix and found the values of true negatives, false positives, false negatives and true positives for N = 13.

As such I made a KNN model for n = 13 and made a confusion matrix for the same. I used the cross\_val\_predict function that takes the given classifier and creates a prediction matrix using k=5 folds by default.

knn\_clf = KNeighborsClassifier(n\_neighbors=13)

knn\_clf.fit(X, y)

y\_pred = cross\_val\_predict(knn\_clf,X,y)

tn, fp, fn, tp = confusion\_matrix(y,y\_pred).ravel()

print(tn, fp, fn, tp)



### **Question 3:**

I performed Gaussian Naive Bayes Classifier on our dataset using 5-folds. I then tweaked the Var Smoothing hyper parameter between a set of defined powers ranging from 10^-9 uptil 10^-3, I cross validated the GaussianNB classifier similarly as in Question 1 with the scoring hyper parameter set for accuracy, precision, recall and f-score metrics. I then proceeded to store the average value for all folds in each iteration into arrays for both training and testing to plot them for easier understanding of all the performance metrics.

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

# Gaussian Naive Bayes Classifier

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

scoring = ['accuracy', 'precision\_macro', 'recall\_macro', 'f1\_macro']

tr\_acc = []

te\_acc = []

tr\_pre = []

te\_pre = []

tr\_rec = []

te\_rec = []

tr\_f1s = []

te\_f1s = []

print("Gaussian Naive Bayes Classifier:")

steps = [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3]

for step in steps:

gnb\_clf = GaussianNB(var\_smoothing=step)

gnb\_clf.fit(X, y)

cv\_results = cross\_validate(gnb\_clf, X, y, cv=5, scoring=scoring, return\_train\_score=True)

tr\_acc.append(np.average(cv\_results['train\_accuracy']))

te\_acc.append(np.average(cv\_results['test\_accuracy']))

tr\_pre.append(np.average(cv\_results['train\_precision\_macro']))

te\_pre.append(np.average(cv\_results['test\_precision\_macro']))

tr\_rec.append(np.average(cv\_results['train\_recall\_macro']))

te\_rec.append(np.average(cv\_results['test\_recall\_macro']))

tr\_f1s.append(np.average(cv\_results['train\_f1\_macro']))

te\_f1s.append(np.average(cv\_results['test\_f1\_macro']))

plt.figure(5)

plt.plot(steps, tr\_acc, label='Training')

plt.plot(steps, te\_acc, label='Testing')

plt.xlabel("Var Smoothing")

plt.ylabel("Accuracy")

plt.legend()

plt.figure(6)

plt.plot(steps, tr\_pre, label='Training')

plt.plot(steps, te\_pre, label='Testing')

plt.xlabel("Var Smoothing")

plt.ylabel("Precision")

plt.legend()

plt.figure(7)

plt.plot(steps, tr\_rec, label='Training')

plt.plot(steps, te\_rec, label='Testing')

plt.xlabel("Var Smoothing")

plt.ylabel("Recall")

plt.legend()

plt.figure(8)

plt.plot(steps, tr\_f1s, label='Training')

plt.plot(steps, te\_f1s, label='Testing')

plt.xlabel("Var Smoothing")

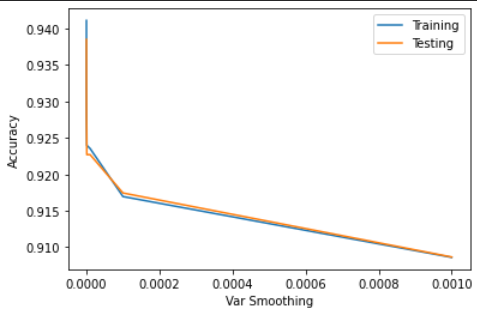
plt.ylabel("F Score")

plt.legend()

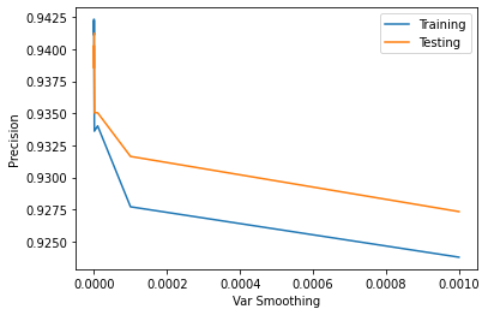
#### **Question 4:**

As discussed in Answer 3, I calculated and constructed plots for the training accuracy, testing accuracy, recall, precision, F-score, and then found the true negatives, false positives, false negatives and true positives using the confusion matrix.

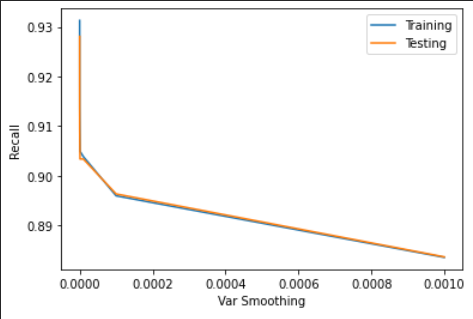
**Average Accuracy vs Var Smoothing**

****

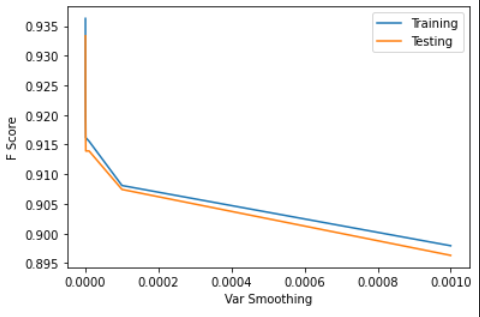
**Average Precision vs Var Smoothing**

****

**Average Recall vs Var Smoothing**

****

**Average F-Score vs Var Smoothing**

****

Upon observing, the plots seem to show that the training and testing performance metrics all bear very close resemblance to each other except for the precision metric. This shows that the var smoothing hyper parameter substantially affects the training data model precision as the value for the training precision is much lower than that of the testing precision in all values of var smoothing. This is due to the var smoothing allowing more data to be misinterpreted in an attempt to smoothen the prediction ending up in overfitting most of the data. This also shows that training performance may not necessarily lead to a better, more well-fit model.

I made a confusion matrix using the default hyper parameters for the Gaussian Naive Bayes Classifier to observe how it would perform in comparison.

gnb\_clf = GaussianNB()

gnb\_clf.fit(X, y)

y\_pred = cross\_val\_predict(gnb\_clf, X, y)

tn, fp, fn, tp = confusion\_matrix(y,y\_pred).ravel()

print(tn, fp, fn, tp)



##### **Question 5:**

Interestingly, the values for the true negatives, false positives, false negatives and true positives I obtained from both the confusion matrices for the KNN and Naive Bayes Algorithm are very close to one another. To further evaluate this, I printed the averages for performance metrics for the KNN and Naive Bayes Classifiers obtained in Question 2 and 4 respectively. The values obtained are very close to one another. This leads to my conclusion of the model being well fitted for n=13 for the KNN model and default hyper parameters being prior=[357/569=0.6274, 212/569=0.3726] (which can be calculated from the dataset’s class distribution: 357 benign, 212 malignant), var smoothing=10^-9 for the Gaussian Naive Bayes Model.

cv\_results = cross\_validate(knn\_clf, X, y, cv=5, scoring=scoring, return\_train\_score=True)

print(np.average(cv\_results['train\_accuracy']))

print(np.average(cv\_results['test\_accuracy']))

print(np.average(cv\_results['train\_precision\_macro']))

print(np.average(cv\_results['test\_precision\_macro']))

print(np.average(cv\_results['train\_recall\_macro']))

print(np.average(cv\_results['test\_recall\_macro']))

print(np.average(cv\_results['train\_f1\_macro']))

print(np.average(cv\_results['test\_f1\_macro']))

|  |  |  |  |
| --- | --- | --- | --- |
| cv\_results = cross\_validate(gnb\_clf, X, y, cv=5, scoring=scoring, return\_train\_score=True)  print(np.average(cv\_results['train\_accuracy']))  print(np.average(cv\_results['test\_accuracy']))  print(np.average(cv\_results['train\_precision\_macro']))  print(np.average(cv\_results['test\_precision\_macro']))  print(np.average(cv\_results['train\_recall\_macro']))  print(np.average(cv\_results['test\_recall\_macro']))  print(np.average(cv\_results['train\_f1\_macro']))  print(np.average(cv\_results['test\_f1\_macro'])) | | | |
|  | | | |