**Homework 3**

**Problem 1**

1. Use SVM with linear kernel to construct your classifier.
   1. Split the data set into training and test sets. Use the partition obtained to answer all questions of problem. This step should be done once and used on the different classifiers. This means that you should use the same training and test sets to construct your classifiers. Hence, it will be easier to compare the classifiers.

To split the data set into training and testing sets I first imported the data using the url of the file. I then imported train\_test\_split from sklearn. I then split the data into target and predictor train and testing sets. See code below:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import auc, roc\_curve

from sklearn import datasets

from sklearn.metrics import confusion\_matrix, classification\_report, precision\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.svm import SVC

from sklearn import metrics

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/sonar.all-data"

data = pd.read\_csv(url)

data.head()

data

df = pd.DataFrame(data)

X = df.iloc[:,0:60]

X.head()

y = df.iloc[:,60]

y.head()

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.25, random\_state=0)

* 1. Use k-fold cross validation on the training set to select the best value(s)of the tuning parameter(s) needed.

I used GridSearchCV from sklearn in order to select the best value of the tuning parameter. After first testing C range 1, 100 with steps 1 and gamma from 0.01 to 0.05 I found that 54 was the best value for C and 0.01 was the best value for Gamma. I did further tests of range for Gamma however, 0.01 produced the best accuracy score. See code below:

c\_range = np.arange(50,60,0.5)

gamma\_range = np.arange(0.01,0.05,0.01)

tuned\_parameters = [{'C': c\_range,

'gamma': gamma\_range}]

clf = GridSearchCV(SVC(kernel='linear'), tuned\_parameters, cv=10, scoring='accuracy')

clf.fit(X\_train, y\_train)

print(clf.best\_params\_)

print(clf.best\_estimator\_.score(X\_test, y\_test))

Results:

{'C': 54, 'gamma': 0.01}  
0.7692307692307693

* 1. Report the performance of each model on the training set, by training misclassification error rate.
  2. Report the confusion matrix. Use the confusion matrix to find ‘Recall, Specificity, Fallout, Positive predictive value, and the accuracy’ for the training and test sets.

I did these two parts together since they are closely related. Using the tuned parameters I fit the training data. I then used the svm.predict commands to perform the predictions. From this I was able to produce a confusion matrix and using these results were able to calculate the corresponding metrics. See code below:

svm = SVC(C=54, kernel='linear', gamma=0.01, probability=True)

svm.fit(X\_train, y\_train)

y\_pred\_linear\_train= svm.predict(X\_train)

print(classification\_report(y\_train, y\_pred\_linear\_train, digits=3))

print(confusion\_matrix(y\_train, clf.best\_estimator\_.predict(X\_train)))

conf\_l = pd.DataFrame(confusion\_matrix(y\_train, clf.best\_estimator\_.predict(X\_train)))

TPl = conf\_l.iloc[0,0]

FPl = conf\_l.iloc[1,0]

FNl = conf\_l.iloc[0,1]

TNl = conf\_l.iloc[1,1]

print("Recall:", TPl/(TPl+FNl))

print("Specificity:", TNl/(TNl+FPl))

print("Fallout:", FPl/(FPl + TNl))

print("PPV:", TPl/(TPl + FPl))

print("Accuracy:", (TPl + TNl)/(TPl +TNl + FPl + FNl))

print("Misclassification Rate:", 1 - ((TPl + TNl)/(TPl +TNl + FPl + FNl)))

Results:

[[76  8]  
 [ 5 66]]

Recall: 0.9047619047619048  
Specificity: 0.9295774647887324  
Fallout: 0.07042253521126761  
PPV: 0.9382716049382716  
Accuracy: 0.9161290322580645  
Misclassification Rate: 0.08387096774193548

* 1. Report the performance of each model on the test set, by reporting the test misclassification error rate.

I repeated the previous process on the testing set. See code below:

y\_pred\_linear\_test = svm.predict(X\_test)

print(classification\_report(y\_test, y\_pred\_linear\_test, digits=3))

print(confusion\_matrix(y\_test, clf.best\_estimator\_.predict(X\_test)))

conf\_l1 = pd.DataFrame(confusion\_matrix(y\_test, clf.best\_estimator\_.predict(X\_test)))

TPl1 = conf\_l1.iloc[0,0]

FPl1 = conf\_l1.iloc[1,0]

FNl1 = conf\_l1.iloc[0,1]

TNl1 = conf\_l1.iloc[1,1]

print("Recall:", TPl1/(TPl1+FNl1))

print("Specificity:", TNl1/(TNl1+FPl1))

print("Fallout:", FPl1/(FPl1 + TNl1))

print("PPV:", TPl1/(TPl1 + FPl1))

print("Accuracy:", (TPl1 + TNl1)/(TPl1 +TNl1 + FPl1 + FNl1))

print("Misclassification Rate:", 1 - ((TPl1 + TNl1)/(TPl1 +TNl1 + FPl1 + FNl1)))

Results:

(omitted other prints as they are not a part of this question)

Misclassification Rate: 0.23076923076923073

1. Use SVM with polynomial kernel to construct your classifier.
   1. Use k-fold cross validation on the training set to select the best value(s)of the tuning parameter(s) needed.

I repeated the process of GridSearchCV. This resulted in a C value of 93 and a gamma value of 0.04. See code below:

c\_range\_poly = np.arange(1,100,1)

gamma\_range\_poly = np.arange(0.01,0.05,0.01)

tuned\_parameters\_poly = [{'C': c\_range\_poly,

'gamma': gamma\_range\_poly}]

clf\_poly = GridSearchCV(SVC(kernel='poly'), tuned\_parameters\_poly, cv=10, scoring='accuracy')

clf\_poly.fit(X\_train, y\_train)

print(clf\_poly.best\_params\_)

print(clf\_poly.best\_estimator\_.score(X\_test, y\_test))

Results:

{'C': 93, 'gamma': 0.04}  
0.7692307692307693

* 1. Report the performance of each model on the training set, by reporting the training misclassification error rate.
  2. Report the confusion matrix. Use the confusion matrix to find ‘Recall, Specificity, Fallout, Positive predictive value, and the accuracy’ for the training and test sets.

Repeated the same process for the linear kernel but replacing kernel=’linear’ to kernel =’poly’ using the new tuned parameters. See code below:

svm\_poly = SVC(C=93, kernel='poly', gamma=0.04, probability=True)

svm\_poly.fit(X\_train, y\_train)

y\_pred\_poly\_train = svm\_poly.predict(X\_train)

print(classification\_report(y\_train, y\_pred\_poly\_train, digits=3))

print(confusion\_matrix(y\_train, clf\_poly.best\_estimator\_.predict(X\_train)))

conf\_poly = pd.DataFrame(confusion\_matrix(y\_train, clf\_poly.best\_estimator\_.predict(X\_train)))

TP\_poly = conf\_poly.iloc[0,0]

FP\_poly = conf\_poly.iloc[1,0]

FN\_poly = conf\_poly.iloc[0,1]

TN\_poly = conf\_poly.iloc[1,1]

print("Recall:", TP\_poly/(TP\_poly+FN\_poly))

print("Specificity:", TN\_poly/(TN\_poly+FP\_poly))

print("Fallout:", FP\_poly/(FP\_poly + TN\_poly))

print("PPV:", TP\_poly/(TP\_poly + FP\_poly))

print("Accuracy:", (TP\_poly + TN\_poly)/(TP\_poly +TN\_poly + FP\_poly + FN\_poly))

print("Misclassification Rate:", 1 - ((TP\_poly + TN\_poly)/(TP\_poly +TN\_poly + FP\_poly + FN\_poly)))

Results:

[[78  6]  
 [ 4 67]]  
Recall: 0.9285714285714286  
Specificity: 0.9436619718309859  
Fallout: 0.056338028169014086  
PPV: 0.9512195121951219  
Accuracy: 0.9354838709677419  
Misclassification Rate: 0.06451612903225812

* 1. Report the performance of each model on the test set, by reporting the test misclassification error rate.

Repeating the same process for the test set. See code below:

y\_pred\_poly\_test = svm\_poly.predict(X\_test)

print(classification\_report(y\_test, y\_pred\_poly\_test, digits=3))

print(confusion\_matrix(y\_test, clf\_poly.best\_estimator\_.predict(X\_test)))

conf\_\_poly1 = pd.DataFrame(confusion\_matrix(y\_test, clf\_poly.best\_estimator\_.predict(X\_test)))

TP\_poly1 = conf\_\_poly1.iloc[0,0]

FP\_poly1 = conf\_\_poly1.iloc[1,0]

FN\_poly1 = conf\_\_poly1.iloc[0,1]

TN\_poly1 = conf\_\_poly1.iloc[1,1]

print("Recall:", TP\_poly1/(TP\_poly1+FN\_poly1))

print("Specificity:", TN\_poly1/(TN\_poly1+FP\_poly1))

print("Fallout:", FP\_poly1/(FP\_poly1 + TN\_poly1))

print("PPV:", TP\_poly1/(TP\_poly1 + FP\_poly1))

print("Accuracy:", (TP\_poly1 + TN\_poly1)/(TP\_poly1 +TN\_poly1 + FP\_poly1 + FN\_poly1))

print("Misclassification Rate:", 1 - ((TP\_poly1 + TN\_poly1)/(TP\_poly1 +TN\_poly1 + FP\_poly1 + FN\_poly1)))

Results:

Misclassification Rate: 0.23076923076923073

1. Use SVM with Gaussian (RBF) kernel to construct your classifier.
   1. Use k-fold cross validation on the training set to select the best value(s)of the tuning parameter(s) needed.

I repeated the previous processes using GridSearchCV. This resulted in a C value of 27 ad a gamma value of 0.04. See code below:

c\_range\_rbf = np.arange(1,100,1)

gamma\_range\_rbf = np.arange(0.01,0.05,0.01)

tuned\_parameters\_rbf = [{'C': c\_range\_rbf,

'gamma': gamma\_range\_rbf}]

clf\_rbf = GridSearchCV(SVC(kernel='rbf'), tuned\_parameters\_rbf, cv=10, scoring='accuracy')

clf\_rbf.fit(X\_train, y\_train)

print(clf\_rbf.best\_params\_)

print(clf\_rbf.best\_estimator\_.score(X\_test, y\_test))

Results:

{'C': 27, 'gamma': 0.04}  
0.7692307692307693

* 1. Report the performance of each model on the training set, by reporting the training misclassification error rate.
  2. Report the confusion matrix. Use the confusion matrix to find ‘Recall, Specificity, Fallout, Positive predictive value, and the accuracy’ for the training and test sets.

I repeated the SVC process replacing kernel =’poly’ to kernel =’rbf’ and using the rbf tuned parameters. See code below:

svm\_rbf = SVC(C=27, kernel='rbf', gamma=0.04, probability = True)

svm\_rbf.fit(X\_train, y\_train)

y\_pred\_rbf\_train = svm\_rbf.predict(X\_train)

print(classification\_report(y\_train, y\_pred\_rbf\_train, digits=3))

print(confusion\_matrix(y\_train, clf\_rbf.best\_estimator\_.predict(X\_train)))

conf\_rbf = pd.DataFrame(confusion\_matrix(y\_train, clf\_rbf.best\_estimator\_.predict(X\_train)))

TP\_rbf = conf\_rbf.iloc[0,0]

FP\_rbf = conf\_rbf.iloc[1,0]

FN\_rbf = conf\_rbf.iloc[0,1]

TN\_rbf = conf\_rbf.iloc[1,1]

print("Recall:", TP\_rbf/(TP\_rbf+FN\_rbf))

print("Specificity:", TN\_rbf/(TN\_rbf+FP\_rbf))

print("Fallout:", FP\_rbf/(FP\_rbf + TN\_rbf))

print("PPV:", TP\_rbf/(TP\_rbf + FP\_rbf))

print("Accuracy:", (TP\_rbf + TN\_rbf)/(TP\_rbf +TN\_rbf + FP\_rbf + FN\_rbf))

print("Misclassification Rate:", 1 - ((TP\_rbf + TN\_rbf)/(TP\_rbf +TN\_rbf + FP\_rbf + FN\_rbf)))

Results:

[[81  3]  
 [11 60]]  
Recall: 0.9642857142857143  
Specificity: 0.8450704225352113  
Fallout: 0.15492957746478872  
PPV: 0.8804347826086957  
Accuracy: 0.9096774193548387  
Misclassification Rate: 0.0903225806451613

* 1. Report the performance of each model on the test set, by reporting the test misclassification error rate.

Repeating the same process on the test set. See code below:

y\_pred\_rbf\_test = svm\_rbf.predict(X\_test)

print(classification\_report(y\_test, y\_pred\_rbf\_test, digits=3))

print(confusion\_matrix(y\_test, clf\_rbf.best\_estimator\_.predict(X\_test)))

conf\_\_rbf1 = pd.DataFrame(confusion\_matrix(y\_test, clf\_rbf.best\_estimator\_.predict(X\_test)))

TP\_rbf1 = conf\_\_rbf1.iloc[0,0]

FP\_rbf1 = conf\_\_rbf1.iloc[1,0]

FN\_rbf1 = conf\_\_rbf1.iloc[0,1]

TN\_poly1 = conf\_\_rbf1.iloc[1,1]

print("Recall:", TP\_rbf1/(TP\_rbf1+FN\_rbf1))

print("Specificity:", TN\_rbf1/(TN\_rbf1+FP\_rbf1))

print("Fallout:", FP\_rbf1/(FP\_rbf1 + TN\_rbf1))

print("PPV:", TP\_rbf1/(TP\_rbf1 + FP\_rbf1))

print("Accuracy:", (TP\_rbf1 + TN\_rbf1)/(TP\_rbf1 +TN\_rbf1 + FP\_rbf1 + FN\_rbf1))

print("Misclassification Rate:", 1 - ((TP\_rbf1 + TN\_rbf1)/(TP\_rbf1 +TN\_rbf1 + FP\_rbf1 + FN\_rbf1)))

Results:

Misclassification Rate: 0.23076923076923073

1. Compare the 3 kernels by using ROC curves and AUC.

First step was to convert the classifications from letters to numeric values. To do this I used the replace command and replaced the class “R” to value 1 and class “M” to value 0. I then used the predict\_proba attribute of the svm.predict command to calculate the probability of a correct classification at each observation. I then was able to use the roc\_curve function from sklearn to produce the corresponding false positive rate and true positive rate arrays used to graph the ROC/AUC curve. Once these values were correctly calculated I plotted them to compare. See code below:

y\_test\_binary = y\_test.replace('R', 1)

y\_test\_binary = y\_test\_binary.replace('M', 0)

#linear kernel

y\_pred\_linear\_prob = svm.predict\_proba(X\_test)

fpr\_lr, tpr\_lr, thresholds = roc\_curve(y\_test\_binary, y\_pred\_linear\_prob[: , 1], pos\_label= 1)

print("SVMClassifier (linear kernel): {0}".format(auc(fpr\_lr,tpr\_lr)))

#polynomial kernel

y\_pred\_poly\_prob = svm\_poly.predict\_proba(X\_test)

fpr\_poly, tpr\_poly, thresholds\_poly = roc\_curve(y\_test\_binary, y\_pred\_poly\_prob[:,1], pos\_label= 1)

print("SVMClassifier (polynomial kernel): {0}".format(auc(fpr\_poly,tpr\_poly)))

#rbf kernel

y\_pred\_rbf\_prob = svm\_rbf.predict\_proba(X\_test)

fpr\_rbf, tpr\_rbf, thresholds\_rbf = roc\_curve(y\_test\_binary, y\_pred\_rbf\_prob[:,1], pos\_label= 1)

print("SVMClassifier (rbf kernel): {0}".format(auc(fpr\_rbf,tpr\_rbf)))

# Plot ROC curve

fig = plt.figure(figsize=(12,10))

ax = fig.add\_subplot(111)

# Connect diagonals

ax.plot([0, 1], [0, 1], ls="--")

# Labels

ax.set\_xlabel('False Positive Rate')

ax.set\_ylabel('True Positive Rate')

ax.set\_title('ROC curve')

# Set graph limits

ax.set\_xlim([0.0, 1.0])

ax.set\_ylim([0.0, 1.0])

# Plot each graph

ax.plot(fpr\_lr,tpr\_lr,label = 'linear')

ax.plot(fpr\_poly,tpr\_poly,label='poly')

ax.plot(fpr\_rbf,tpr\_rbf,label='rbf')

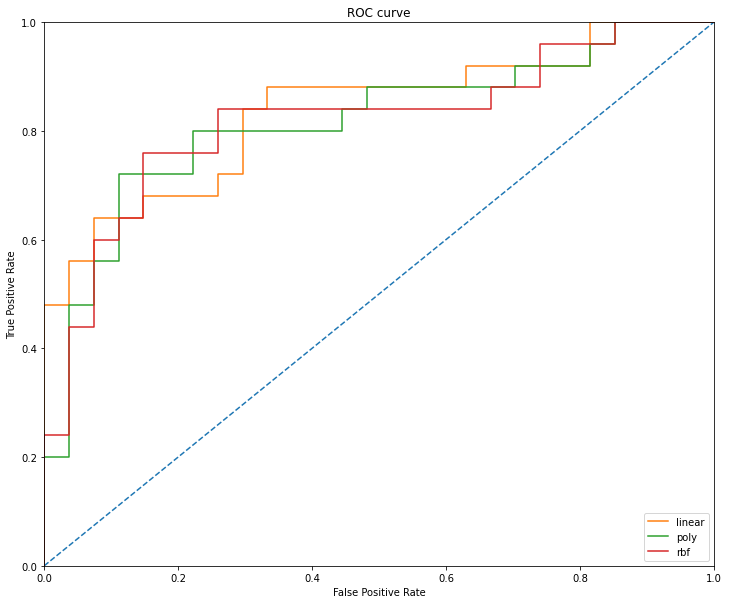
# Set legend and show plot

ax.legend(loc="lower right")

plt.show()

Results:

SVMClassifier (linear kernel): 0.8355555555555555  
SVMClassifier (polynomial kernel): 0.8162962962962963  
SVMClassifier (rbf kernel): 0.8177777777777778



1. Using your the classifiers from the previous questions above, report the best classifier for this dataset. The best classifier is the one with the smallest training and test misclassification errors.

Based off of the misclassification errors. The polynomial kernel has the lowest average between its training/testing misclassification errors.

Linear avg score: 0.15732

Polynomial avg score: 0.147643

RBF avg score: 0.160546