DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Homework Assignment No. 9:

HW No. 9: Bootstrapping, Bagging, and Combining Classifiers

submitted to:

Professor Joseph Picone
ECE 8527: Introduction to Pattern Recognition and Machine Learning
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

March 30th, 2023

prepared by:

Gavin Koma
Email: gavintkoma@temple.edu

A. TASK 1

Step one of this assignment was to alter our dataset 10. This step was fairly straight forward and can be shown in the following code:

```
#%% dataprep
#use the entire training data set
conventional_train = pd.read_csv("00_train_03.csv",header=None)
conventional_dev = pd.read_csv("00_dev_03.csv",header=None)
conventional_eval = pd.read_csv("00_eval_03.csv",header=None)

#before using pca we whould train the model and use a logistic
#regression to just see how well it performs
x_train = conventional_train.iloc[:,1:3]
y_train = conventional_train.iloc[:,0]
x_dev = conventional_dev.iloc[:,1:3]
y_dev = conventional_dev.iloc[:,0]
x_eval = conventional_eval.iloc[:,1:3]
y_eval = conventional_eval.iloc[:,0]


#for bag we need random 75% of dataset
conventional_train_shuffle = conventional_train.sample(frac=1)
conv_train_75 = conventional_train_shuffle.iloc[0:75000,:]
x_75_train = conv_train_75.iloc[:,1:3]
y_75_train = conv_train_75.iloc[:,0]

#bagging method
LR = LogisticRegression(random_state=0)

bag = BaggingClassifier(base_estimator=LR,
                        n_estimators=10,
                        random_state=0)

#this bag is 75% of the original dataset split into 10
bag.fit(x_75_train,y_75_train)



#CV method
k_folds = KFold(n_splits=10)
scores = cross_val_score(LR, x_train, y_train, cv = k_folds)
```

This code is used to show how our data should look in the future when we pass it through the algorithm. For conventional use, we use the entire dataset, for the bagging use, we use 75% of our dataset, and for the cross-validation method (CV) we use the whole training set but split it into 10 separate subsets that are all mutually exclusive.

B. TASK 2

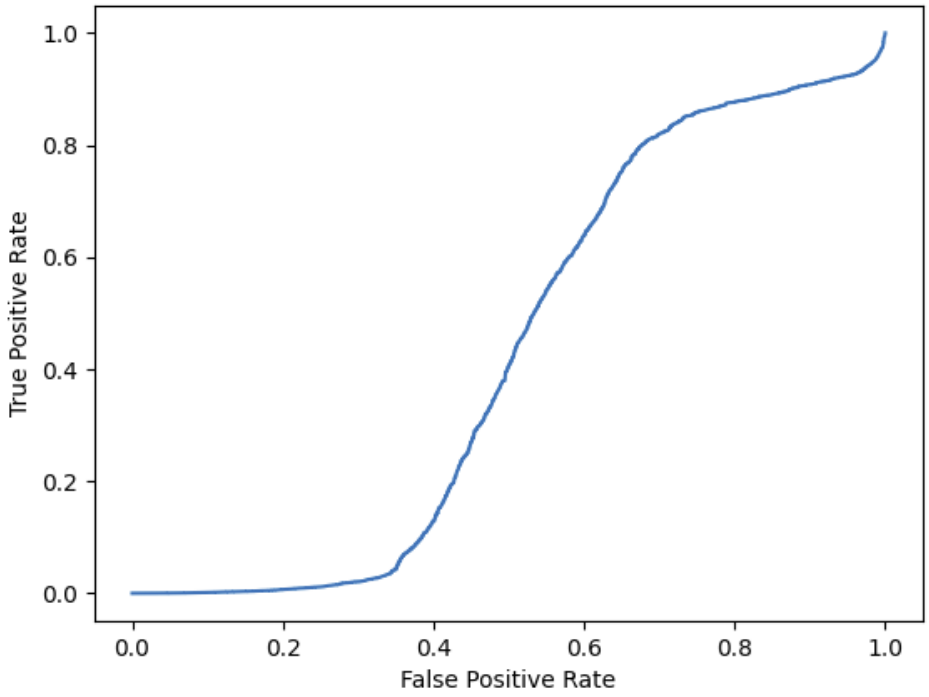
We then use these same methods when performing our algorithmic analyses. For the first one, we are to use a conventional baseline PCA system that is trained on the entire dataset and is evaluate on /eval. At the end of this document, a table is provided that shows the corresponding error rate for each system when performed on /train, /dev, and /eval.

The first step requires us to fit our PCA model, and then transform each of the sets that we plan to use it on. For this homework, we are given the opportunity to use any algorithm and for simplicity's sake, I have chosen to do a simple logistic regression for each task. We initialize the logistic regression model, pass our PCA transformed data to it, and then assess the accuracy (or error rate by subtracting accuracy from 1), and create a plot that shows the ROC curve for A.

The following code was utilized for this task:

```
#%%(A) conventional --> baseline pca system
pca = PCA(n_components=(2))
pca.fit(x_train,y_train)

y_train_pca = y_train
x_train_pca = pca.fit_transform(x_train)

x_dev_pca = pca.fit_transform(x_dev)
x_eval_pca = pca.fit_transform(x_eval)

logreg = LogisticRegression()
logreg.fit(x_train_pca, y_train_pca)
ypred_train = logreg.predict(x_train_pca)
ypred_dev = logreg.predict(x_dev_pca)
ypred_eval = logreg.predict(x_eval_pca)

acc_log_train = logreg.score(x_train_pca, y_train_pca)
print(1-acc_log_train)

acc_log_dev = logreg.score(x_dev_pca,y_dev)
print(1-acc_log_dev)

acc_log_eval = logreg.score(x_eval_pca,y_eval)
print(1-acc_log_eval)

#done with pca

y_pred_proba = logreg.predict_proba(x_train)[::,1]
fpr,tpr,_ = metrics.roc_curve(y_train, y_pred_proba)
plt.plot(fpr,tpr)
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
```

Which outputs this graph:



C. TASK 3

For this task, we are to use conventional bootstrapping. We want to use 9 of the 10 subsets and evaluate on the held-out set. I am unsure if this was done properly but I was able to find a similar article regarding this for combining bootstrapping with a logistic regression analysis. In order to do so, I created a variable that stated how many iterations we wanted the bootstrap to undergo. Since we have 100,000 datapoints in /train, we want each subset to have 10,000 data values and have it evaluate on the last 10,000.

Thus, I created a my logistic regression and specified my iterations; I then passed my iterative values to my algorithm for resampling and used this to calculate my error rate.

```
# train model
reg = LogisticRegression(random_state=0)
reg.fit(x_train, y_train)
n_iterations = 10000

# bootstrap predictions
accuracy_train = []
for i in range(n_iterations):
    X_bs, y_bs = resample(x_train, y_train, replace=True)
    # make predictions
    y_hat = reg.predict(X_bs)
    # evaluate model
    score = accuracy_score(y_bs, y_hat)
    accuracy_train.append(score)
print(1-sum(accuracy_train)/len(accuracy_train))

accuracy_dev = []
for i in range(n_iterations):
    X_bs, y_bs = resample(x_train, y_train, replace=True)
    # make predictions
    y_hat = reg.predict(X_bs)
    # evaluate model
    score = accuracy_score(y_bs, y_hat)
    accuracy_dev.append(score)
print(1-sum(accuracy_dev)/len(accuracy_dev))

accuracy_eval = []
for i in range(n_iterations):
    X_bs, y_bs = resample(x_train, y_train, replace=True)
    # make predictions
    y_hat = reg.predict(X_bs)
    # evaluate model
    score = accuracy_score(y_bs, y_hat)
    accuracy_eval.append(score)
print(1-sum(accuracy_eval)/len(accuracy_eval))
```

D. TASK 4

For this task, we want to utilize a conventional dataset and perform cross-validation on 9 of the 10 sets and perform an evaluation on the last set. In order to do this, initiated my cross-validation command which specifies 10 splits total and would shuffle my data. I then called my model and passed my model through the cross-validation command and calculated my accuracy score from it.

```
#use the entire training data set
conventional_train = pd.read_csv("00_train_03.csv",header=None)
conventional_dev = pd.read_csv("00_dev_03.csv",header=None)
conventional_eval = pd.read_csv("00_eval_03.csv",header=None)

#before using pca we whould train the model and use a logistic
#regression to just see how well it performs
x_train = conventional_train.iloc[:,1:3]
y_train = conventional_train.iloc[:,0]
x_dev = conventional_dev.iloc[:,1:3]
y_dev = conventional_dev.iloc[:,0]
x_eval = conventional_eval.iloc[:,1:3]
y_eval = conventional_eval.iloc[:,0]


# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
model = LogisticRegression()
model.fit(x_train,y_train)
# evaluate model
scores = cross_val_score(model,
                         x_train,
                         y_train,
                         scoring='accuracy',
                         cv=cv,
                         n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))

scores = cross_val_score(model,
                         x_dev,
                         y_eval,
                         scoring='accuracy',
                         cv=cv,
                         n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))


scores = cross_val_score(model,
                         x_eval,
                         y_eval,
                         scoring='accuracy',
                         cv=cv,
                         n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

E. TASK 5

In order to complete this task, we are required to bootstrap our logistic regression. We see that we have to also specify iterations again for this task. In a similar manner to the previous section, we first call our bootstrapping command then proceed to pass the data to our previously specified algorithm and then calculate the score. Here, we average the mean score from each section. To do so, we create an empty list and append our values to that list.

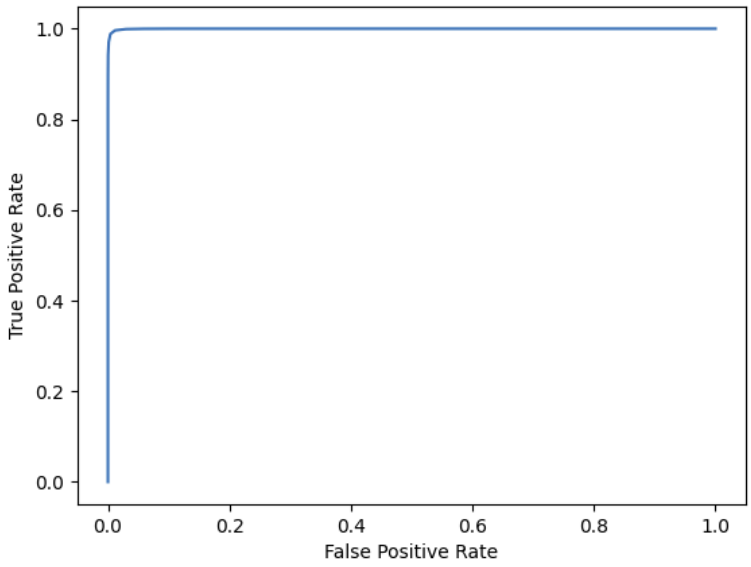
Each value of this list is one mean and it will be used to calculate the final mean across our transformation matrices. Unfortunately, I could not figure out how to include the eval and dev data in one single for-loop. Therefore, I had to poorly code two more for-loops that act the same way as seen below in order to calculate the /dev and /eval error rates for a bootstrapped logistic regression.

```
#use the entire training data set
conventional_train = pd.read_csv("00_train_03.csv",header=None)
conventional_dev = pd.read_csv("00_dev_03.csv",header=None)
conventional_eval = pd.read_csv("00_eval_03.csv",header=None)

#before using pca we whould train the model and use a logistic
#regression to just see how well it performs
x_train = conventional_train.iloc[:,1:3]
y_train = conventional_train.iloc[:,0]
x_dev = conventional_dev.iloc[:,1:3]
y_dev = conventional_dev.iloc[:,0]
x_eval = conventional_eval.iloc[:,1:3]
y_eval = conventional_eval.iloc[:,0]

# Define the number of bootstrap iterations
n_iterations = 1000

# Define a list to store the performance scores
scores = []

# Perform bootstrapping
for i in range(n_iterations):
    # Generate a bootstrap sample
    x_train_boot, y_train_boot = resample(x_train, y_train)

    # Train a binary classifier on the bootstrap sample
    lr = LogisticRegression()
    lr.fit(x_train,y_train)

    # Evaluate the performance of the classifier on the test set
    y_pred = lr.predict(x_train)
    score = accuracy_score(y_train, y_pred)

    # Store the performance score
    scores.append(score)

# Calculate the mean and standard deviation of the performance scores
mean_score = np.mean(scores)
std_score = np.std(scores)

# Print the results
print("train accuracy: {:.4f} +/- {:.4f}".format(mean_score, std_score))
```

F. TASK 6

Our next goal is to utilize bagging and train 10 systems, one on each subset, and then evaluate it by using a majority rule. Similar to last time, we first call the bagging classifier and then use the model to pass our data to our logistic regression and calculate the accuracy score. This one was far more straightforward than the previous ones.

```
# define the model
model = BaggingClassifier()
model.fit(x_75_train,y_75_train)
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)


n_scores = cross_val_score(model,
                           x_75_train,
                           y_75_train,
                           scoring='accuracy',
                           cv=cv,
                           n_jobs=-1,
                           error_score='raise')
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

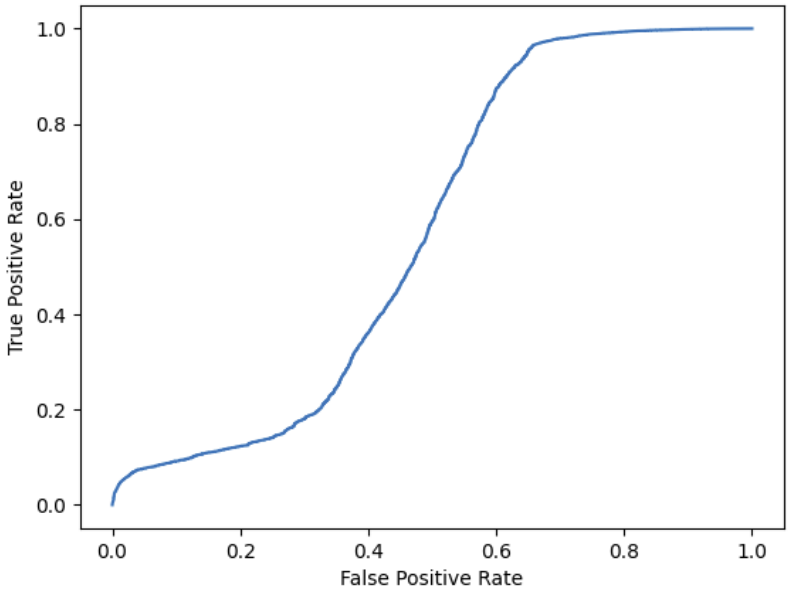


The ROC curves are explained in further detail in the conclusion when they have all been aggregated together.

G. TASK 7

Our final goal is to boost our data and pass it to our logistic regression. As discussed in lecture, we first perform a PCA on the data, then pass the data to our model and use the ada_boost prediction module to perform our predictions. We specify in our AdaBoostClassifier that we are using a logistic regression model and pass other hyperparameters to the model. The following code produces our error rates and the following ROC curve:

```
pca = PCA()
x_train = pca.fit_transform(x_train)
x_eval = pca.fit_transform(x_eval)
x_dev = pca.fit_transform(x_dev)


lr = LogisticRegression()

ada_boost = AdaBoostClassifier(
    base_estimator=lr, n_estimators = 50, random_state=42)

ada_boost.fit(x_train,y_train)

ada_model = ada_boost.fit(x_train,y_train)

y_pred_trainf = ada_boost.predict(x_train)
y_pred_evalf = ada_boost.predict(x_eval)
y_pred_devf = ada_boost.predict(x_dev)

train_error = 1- (metrics.accuracy_score(y_train,y_pred_trainf))
dev_error = 1- (metrics.accuracy_score(y_dev,y_pred_devf))
eval_error = 1- (metrics.accuracy_score(y_eval,y_pred_evalf))

print("train error score is: ",train_error)
print("dev error score is: ",dev_error)
print("eval error score is: ",eval_error)

y_pred_proba = ada_boost.predict_proba(x_75_train)[::,1]
fpr,tpr,_ = metrics.roc_curve(y_75_train, y_pred_proba)
plt.plot(fpr,tpr)
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
```



H. CONCLUSION

Our final goals are to provide an aggregated table of all the error rates and to provide one ROC curve with all the data.

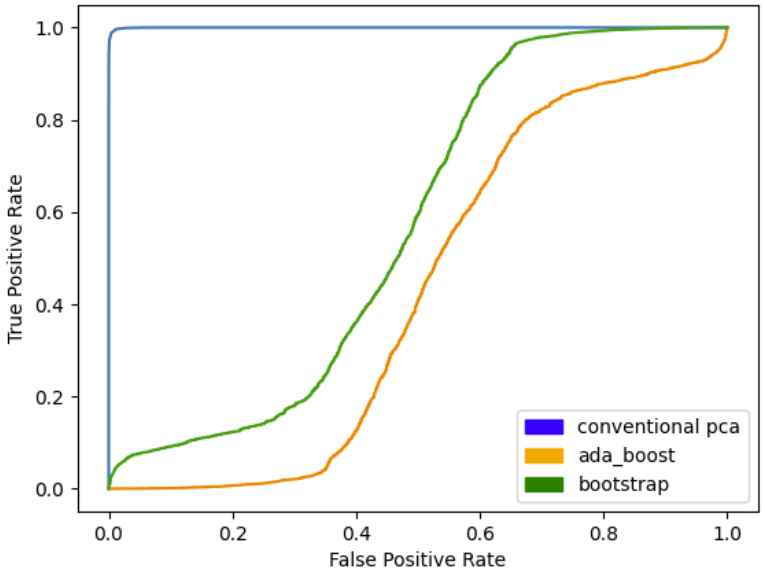
Error Rate of Systems A-F

System	/train	/dev	/eval
A	0.4699	0.4551	0.4871
B	0.4698	0.4699	0.4700
C	0.4699	0.4491	0.5244
D	0.4501	0.4711	0.4201
E	0.0898	0.1339	0.1699
F	0.4722	0.4534	0.4883

All systems seem to provide approximately the same error rate *except* system e which utilizes bagging. After doing some reading, it seems that AdaBoost functions by combining multiple weak classifiers that only really perform better than random guessing to produce one single strong classifier that can work to accurately classify our data. This is seen in the table above.

A hint to this is given in the homework when we are told to train 10 systems, one on each subset and to evaluate using the majority vote; AdaBoost does exactly that. Each weak classifier is trained on a subset of training data and the algorithm focuses on learning the difficult examples in order to improve overall performance.

Unfortunately, I do not see this resembled in my ROC curve, but I cannot figure out what parts of my code are incorrect. Based on the ROC curves, ada_boost performs the worst while conventional PCA performs the best, even though this is most definitely not what is seen in the error rates. I wonder if my ROC curves are obtaining the true positive values and the false positive values from the weak classifiers that are used within adaboost, but I am not able to find any information online that could explain the drastic differences that are seen in my aggregated ROC curve and my aggregated error rate table.



To conclude, I struggled a lot with this assignment. I didn't realize the methods that were needed to include in my code to properly perform this homework and it took an incredibly long time to understand how to incorporate specific techniques. I think that this can only be completed again with practice, and I am happy to have been exposed to it. I know that I will undoubtedly have to do this again in the future and it is great that I have done it in this assignment, but I am not entirely confident in my ability to implement them properly. I am actually looking forward to the final project now to reassess these skills and see how I can implement them to get the best accuracy possible.