CS 484 Homework 4 Report

By: Huda Khalid

Miner Username: whodabest

I chose to implement my homework 4 by using collaborative filtering. Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users. Consider a user C, find set D of other users whose ratings are “similar” to C’s ratings. Then, estimate user’s ratings based on ratings of users in D. I will be using the training and test files/dataset for my implementation.

**Training Set**

It was my first-time using pandas so I had to learn the library and it parameters before I could continue.

I read in the training file using pandas, where it took me forever to understand how to have each column go into their respected place. I had to use the parameter delim\_whitespace = ‘true’ in order for pandas to read in each column and not consider it one whole column by itself. I then used the same approach I used for what we did in homework 2 where we split the training data set in order to train the training data to come up with the best results using the best classifier. I created an X = trainData[[“userID”,”movieID”]] and y = trainData[[“rating”]] in which I saved the respected columns in order to test it. I then went straight to the classifier and defined the X\_train as X and y\_train as y, so on and so forth. The first classifier I decided to test with was KNearestNeighbor in which I was getting an error when trying to fit the model. However, after much research I decided to turn the whole trainData.astype(int) because it wouldn’t allow me to use floats for it being continuous. I went back and reevaluated my implementation and algorithm and decided to use numpy arrays since that’s what I am most comfortable with and we are working with numbers here and not text. Therefore, right after I read in the data using pandas I conver the Training Data and save the columns in their respected place:

trainData = trainData.astype(int)  
trainData = np.array(trainData)  
#print(trainData)  
#userID and movieID  
X = trainData[:**,**:**2**]  
#ratings  
y = trainData[:**,**-**1**]

I then incorporated every classifier we learned about this semester, for example, KNN, NaiveBayes, and decision tree.

The first preprocessing step is to divide the dataset into a feature set and corresponding labels which I have done above. The script above stores the feature sets into the X variable and the series of corresponding labels into the y variable. The next preprocessing step is to divide data into training and test sets. I execute the following script to do so:

folds = KFold(n\_splits=**4**)

X\_train**,** X\_test**,** y\_train**,** y\_test = train\_test\_split(X**,** y**,**test\_size=**0.3**)

I used 4 n\_splits for k-fold cross validation where we divide the dataset into K equal sized sections. The algorithm is tested K times, each time leaving out one of the K sections from building the classifier but using it to test the classifier instead. We divide our dataset into training and test sets, so we can estimate how well the algorithm performs on new data. We will split the data into 70% train data and 30% test data. Cross Validation is a technique which involves reserving a particular sample of a dataset on which you do not train the model. Train the model using the remaining part of the dataset. Use the reserve sample of the test (validation) set.

I created a for loop where I iterate through to perform cross validation and classifier on each fold.

for train**,** test in folds.split(X):  
 X\_train**,** X\_test**,** y\_train**,** y\_test = X[train]**,** X[test]**,** y[train]**,** y[test]  
  
 knn = KNeighborsClassifier(n\_neighbors=**5,**weights='uniform'**,**p=**1**)  
 knn.fit(X\_train**,** y\_train)  
 knnPredict = knn.predict(X\_test)  
 knnArray.append(accuracy\_score(y\_test**,**knnPredict))

DT = DecisionTreeClassifier(class\_weight='balanced')  
DT.fit(X\_train**,** y\_train)  
DTPredict = DT.predict(X\_test)  
dtArray.append(accuracy\_score(y\_test**,** DTPredict))  
  
gausNB = GaussianNB()  
gausNB.fit(X\_train**,** y\_train)  
gausNBPredict = gausNB.predict(X\_test)  
gausNBArray.append(accuracy\_score(y\_test**,** gausNBPredict))

According to my notes and knowledge, the goal of PCA is to reduce the number of dimensions and transfer interdependent variables into single and independent components. PCA transforms the data into a lower dimensional space, by constructing dimensions that are linear combinations of the input dimensions/features. By incorporating PCA on the training set the accuracy doesn’t change much for KNN and Naïve Bayes, however, it does decrease for decision tree.

pca = PCA()  
pca.fit(X\_train)  
pca.fit(X\_test)  
X\_train = pca.transform(X\_train)  
X\_test = pca.transform(X\_test)

The next process we want to try is doing oversampling for imbalanced sets.

oversample = RandomOverSampler()  
X\_train**,** y\_train = oversample.fit\_resample(X\_train**,** y\_train)

By including oversampling, the accuracy scores decreased for every classifier. I am going to now implement both PCA and Oversampling at the same time as I did in homework 2 to see what scores I receive:

Classifying with knn

[2 2 2 ... 1 2 2]

0.23517258116116957

Classifying with decision tree

[3 3 3 ... 0 4 4]

0.23045207590022018

Classifying with Naive Bayes

[5 5 5 ... 4 4 4]

0.1267401084706174

As you can see above, the scores are not all that great! I also used standard scaler for normalizing the data:

scaler = StandardScaler()  
scaler.fit(X\_train)  
X\_train = scaler.transform(X\_train)  
X\_test = scaler.transform(X\_test)

I get better results: .33 for KNN, .27 for DT, .36 for NB. Now it is just the matter of which classifier and best normalization method is used to provide the best results. I tried hyper tuning the parameters of standardScaler but I seem to get the same results. I also tried using standard scaler and PCA and I get the results of .32 with KNN, .22 with DT, and .36 with NB. With StandardScaler and oversampling I get results .24 with KNN, .27 with DT, .13 for NB and as you can tell the results decreased significantly. I came to a conclusion that using oversampling on the train data set produces lower accuracy scores.

I used standard scaler then PCA(n\_components=2) and it produces the results below:

Classifying with knn

[3 3 3 ... 3 2 2]

0.3241831586882264

Classifying with decision tree

[3 3 3 ... 0 4 4]

0.21049260079458595

Classifying with Naive Bayes

[4 4 4 ... 4 4 4]

0.35734822576608016

I also implemented kmeans clustering while doing the kfold cross validation:

kmeans1 = KMeans(n\_clusters=**5**)  
kmeans1.fit(X\_train**,** y\_train)  
predictions = kmeans1.fit\_predict(X\_test)  
clusterArray.append(accuracy\_score(y\_test**,** predictions))

I didn’t get the best results. However, I am going to use all the classifiers on the test set to see which best score I get on miner. Some things might work with the training set and some things might work with the test set. It is all about experiment and choosing the best results.

**Test Set**

I first read in the testing data like how I did for the training set:

testData = pd.read\_csv('/Users/hudakhalid/Documents/CS484/HW4\_khalid/additional\_files/test.dat'**,**delim\_whitespace='true'**,**usecols=["userID"**,**"movieID"])  
testData.head()  
print(testData)  
testData = testData.astype(int)  
testData = np.array(testData)

I then created and initialized the respected variables that I will be using for the testSet

X\_train = X  
y\_train = y  
X\_test = testData  
knnArray = []  
knnPredict = []  
clusterArray = []  
predictions = []  
dtArray = []  
gausNBArray = []

I then go ahead and use one classifier at a time on the test set to see which best score I get on miner. The first runner up for testing is the kmeans clustering implementation:

kmeans1 = KMeans(n\_clusters=**5**)  
kmeans1.fit(X\_train**,** y\_train)  
predictions = kmeans1.fit\_predict(X\_test)

I got an RMSE of 2.55 without using any normalization. I am now going to run it again with using the normalization methods of StandardScaler and dimensionality reduction using PCA which I will be using for all the other classifiers:

scaler = StandardScaler()  
scaler.fit(X\_train)  
X\_train = scaler.transform(X\_train)  
X\_test = scaler.transform(X\_test)  
pca = PCA()  
pca.fit(X\_train)  
pca.fit(X\_test)  
X\_train = pca.transform(X\_train)  
X\_test = pca.transform(X\_test)

With using normalization, the RMSE decreased to a 2.07. I am going to test the next classifier which is KNearestNeighbor without any normalization and dimensionality reduction and I get a score of 1.22 for the RMSE on miner. I am also going to try the KNN algorithm with normalization and dimensionality reduction to see what gives me the better RMSE on miner.

knn = KNeighborsClassifier(n\_neighbors=**5,** weights='uniform'**,** p=**1**)  
knn.fit(X\_train**,** y\_train)  
knnPredict = knn.predict(X\_test)

It is all experimenting with trial and error. I received an RMSE of 1.23 while using PCA and standard Scaler which is not much of a difference.

The next runner up for classifiers is decision tree without any normalizing or dimension reduction:

DT = DecisionTreeClassifier(class\_weight='balanced')  
DT.fit(X\_train**,** y\_train)  
DTPredict = DT.predict(X\_test)

I receive a score of 1.29 for RMSE on miner. I am next going to try the decision tree classifier with standard scaler and PCA. I receive a score of 1.35 RMSE.

gausNB = GaussianNB()  
gausNB.fit(X\_train**,** y\_train)  
gausNBPredict = gausNB.predict(X\_test)

The final classifier I am using is Naïve Bayes without any use of StandardScaler or PCA and I receive an RMSE of 1.11. I also test Naïve Bayes with normalizing the test set and received an RMSE of 1.11 as well.

I also wanted to mention that I am using the code below to print out to a .txt file to submit to miner:

formatFile = open('hkhalidHW4.txt'**,**'w')  
for i in knnPredict:  
 formatFile.write(str(i) + '\n')  
  
formatFile.close()