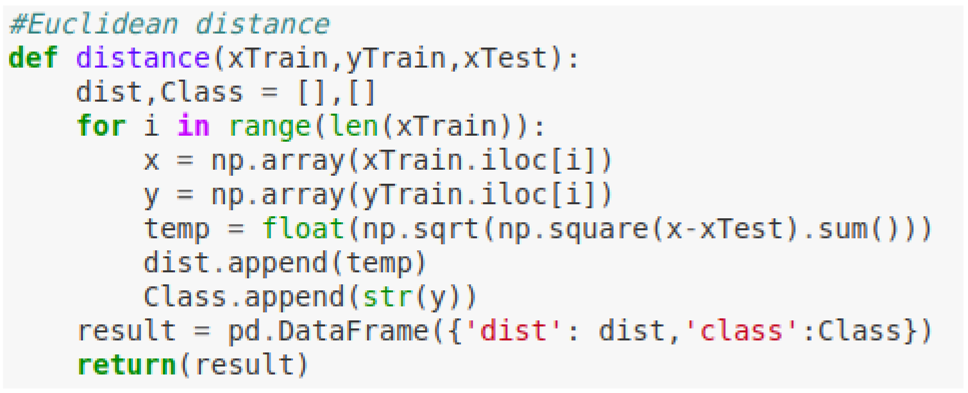
Lab Course Machine Learning
Exercise 7
Krithika Murugesan(277537)

Exercise 1: Implement K-Nearest Neighbor (KNN)

Datasets : Iris Dataset; The dependent variable is the class of the Iris namely, Setosa, Versicolor and Virginica. The other four variables are petal and sepal widths and lengths. There are no null values in the dataset. It is divided into train and test data for learning kNN, the prediction is done by choosing the most frequently occurring class in the k nearest neighbors.

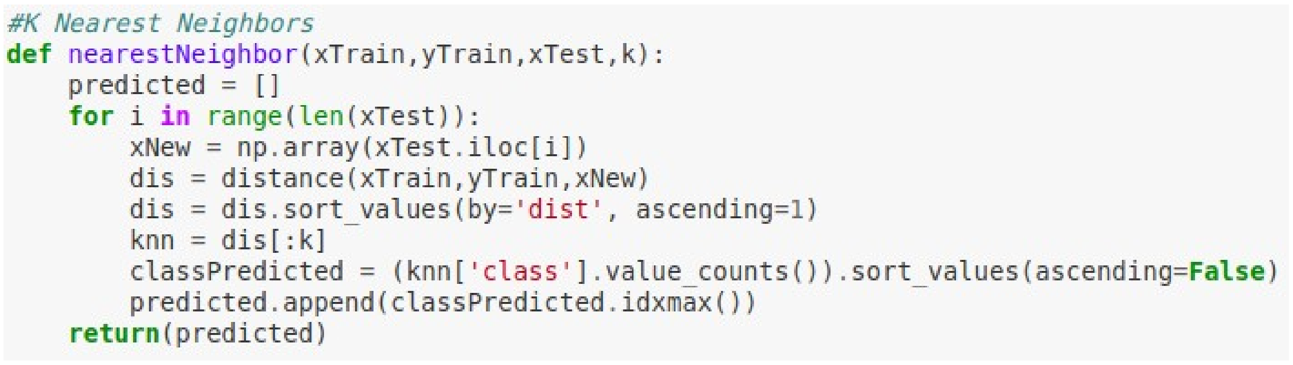
Wine Dataset; The dependent variable is the quality of the wine, there are no anomalies in the data and it is separated into train and test. Considering this to be a regression dataset, the test values are predicted by taking the mean of values of the k nearest neighbors.

The similarity measure computed to obtain the k nearest neighbors is euclidean distance, the following code snippet is used to perform the same. It returns the distance for each test row with all train data, as kNN is a lazy algorithm



```
#Euclidean distance
def distance(xTrain,yTrain,xTest):
    dist,Class = [],[]
    for i in range(len(xTrain)):
        x = np.array(xTrain.iloc[i])
        y = np.array(yTrain.iloc[i])
        temp = float(np.sqrt(np.square(x-xTest).sum()))
        dist.append(temp)
        Class.append(str(y))
    result = pd.DataFrame({'dist': dist,'class':Class})
    return(result)
```

The function that returns the k nearest neighbors is as follows, where we sort the rows by distance and take the top k values with the least distance later performing regression or classification on them based on requirement.



```
#K Nearest Neighbors
def nearestNeighbor(xTrain,yTrain,xTest,k):
    predicted = []
    for i in range(len(xTest)):
        xNew = np.array(xTest.iloc[i])
        dis = distance(xTrain,yTrain,xNew)
        dis = dis.sort_values(by='dist', ascending=1)
        knn = dis[:k]
        classPredicted = (knn['class'].value_counts()).sort_values(ascending=False)
        predicted.append(classPredicted.idxmax())
    return(predicted)
```

For regression lines 8 and 9 are replaced with **predicted.append(knn[‘yValue’].mean())**

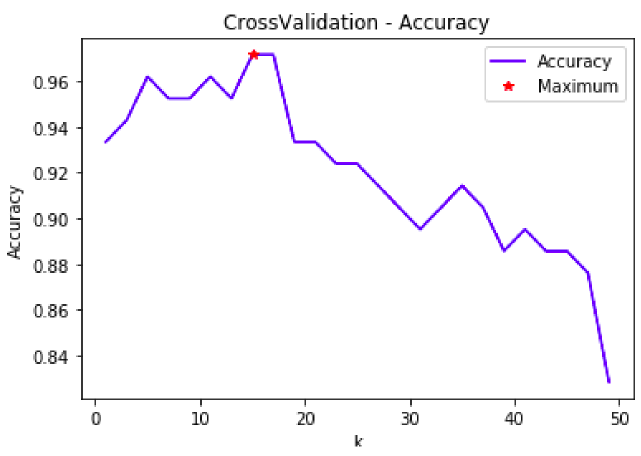
After prediction is made, for classification problems the accuracy and mis-classification rate are used as measures to check the quality of prediction. These are complimentary to each other. Similarly RMSE can be used for Regression problems.

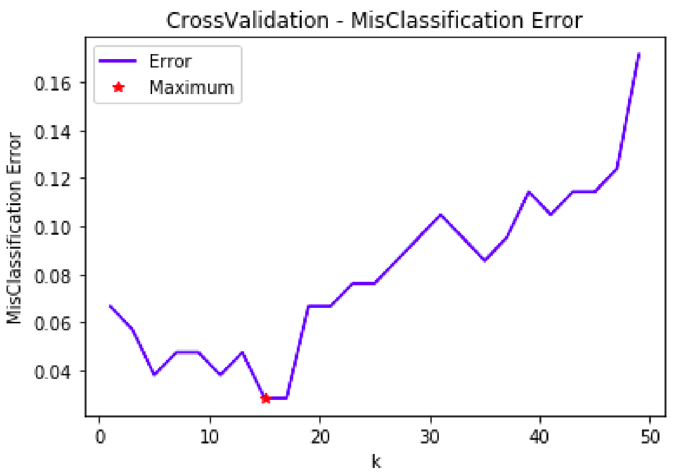
The accuracy for kNN model for Iris Dataset is 0.977, this is a pretty good accuracy considering k=2.

<u>Exercise 2: Optimize and Compare KNN algorithm</u>

PART A: Determine Optimal Value of K in KNN algorithm

k is a hyper parameter and is difficult to find an optimal value using learning algorithm. So cross-validation can be used to compute the best values of k for which the accuracy is high or error is least. Performing cross validation on Iris dataset, it can be seen that the accuracy reaches a maximum at k =15 and then starts decreasing. Therefore 15 is taken as optimal value of k.





It can be seen that the two graphs are inversion of each other as the functions are compliment of each other. Getting the accuracy for different k values directly from the model does not show much variation in accuracy. Below are the cross validation code snippets.

```python
#CrossValidation
def cv(trainCvalid,kGrid,totalFolds):
    k,acc = [],[]
    for eachK in kGrid:
        k.append(eachK)
        eachFoldAccuracy = []
        for each in range(1,totalFolds+1):
            trainC = trainCvalid.copy()
            trainCv,testCv = cvTestTrain(trainC,each,totalFolds)
            xTrain = trainCv.loc[:,testCv.columns != 'class']
            yTrain = trainCv.loc[:,testCv.columns == 'class']
            xTest = testCv.loc[:,testCv.columns != 'class']
            yTest = testCv.loc[:,testCv.columns == 'class']
            predicted = nearestNeighbor(xTrain,yTrain,xTest,eachK)
            accu = accuracy(predicted,yTest)
            eachFoldAccuracy.append(accu)
        temp = (np.mean(eachFoldAccuracy))
        acc.append(float(temp))
    return(k,acc)
```

```
#returns the test and train data for each fold of cross validation
def cvTestTrain(trainA,testFold,totalFolds):
    trainCv = pd.DataFrame([])
    testCv = pd.DataFrame([])
    #Extracting column names
    cols = trainA.columns.values
    #Removing colunm names
    trainA.columns = [''] * len(trainA.columns)
    #Data split to folds
    batch = np.array_split(trainA,totalFolds)
    key = 1
    trainDict = {}
    #Creating the test fold
    for each in batch:
        trainDict[key] = each
        key = key + 1
    testCv = trainDict[testFold]
    testCv.columns = cols
    #Creating train fold
    for key in range(1,6):
        if(key != testFold):
            trainCv = trainCv.append(trainDict[key])
    #Adding the column names to merged train datset
    trainCv.columns = cols
    return(trainCv,testCv)
```

Part B : Compare KNN algorithm with Tree based method

Classification for the same train-test split using kNN and decision trees are done using the SKLEARN packages **KNeighborsClassifier** and **DecionTreeClassifier.** Regressor can be used for regression tasks. Decision trees have several hyper parameters unlike kNN. The following are the code snippets for the same.

```
#Initialize model
knn = KNeighborsClassifier(n_neighbors=3)

#Fit data to model
knn.fit(xTrain, np.array(yTrain).ravel())

#Predicting values for test data
expected = yTest
predicted = knn.predict(xTest)

#Computing accuracy
print("Model summary",metrics.classification_report(expected, predicted))
print("\nConfusion Matrix",metrics.confusion_matrix(expected, predicted))
print ("\nAccuracy",accuracy_score(expected, predicted))
```

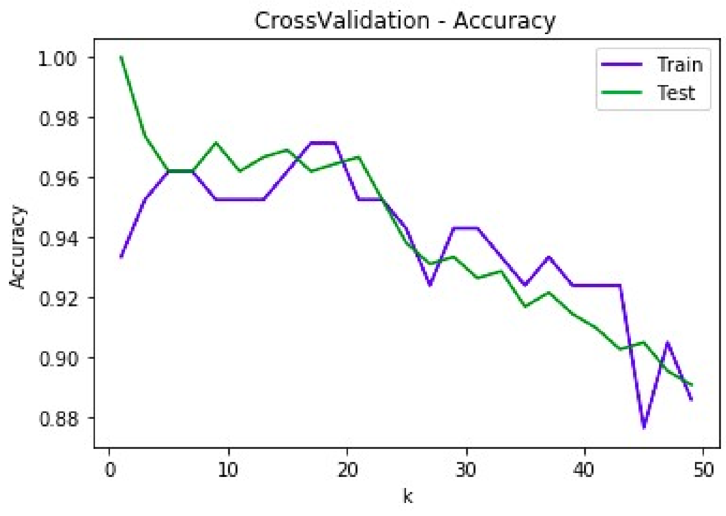
```
Model summary                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        15
Iris-versicolor       1.00      0.94      0.97        16
 Iris-virginica       0.93      1.00      0.97        14

    avg / total       0.98      0.98      0.98        45


Confusion Matrix [[15  0  0]
 [ 0 15  1]
 [ 0  0 14]]

Accuracy 0.977777777778
```

```
#Fit data
model.fit(xTrain, yTrain)
print(model)

# make predictions
expected = yTest
predicted = model.predict(xTest)

# summarize the fit of the model
print("Model summary",metrics.classification_report(expected, predicted))
print("\nConfusion Matrix",metrics.confusion_matrix(expected, predicted))
print("\nAccuracy",accuracy_score(expected, predicted))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
Model summary                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        15
Iris-versicolor       0.94      1.00      0.97        16
 Iris-virginica       1.00      0.93      0.96        14

    avg / total       0.98      0.98      0.98        45


Confusion Matrix [[15  0  0]
 [ 0 16  0]
 [ 0  1 13]]

Accuracy 0.977777777778
```

It can be seen that the accuracy for both the models are almost same, the difference can be spotted in the Confudion matrix, where wrong classifications are made.

To get the optimal values of k, gridsearch cross validation is done, the results can be plotted as

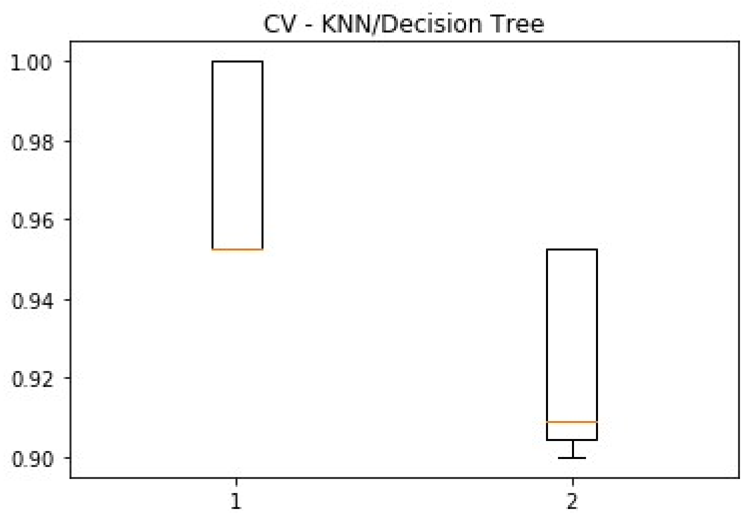


It can be observed that the optimal k value for manually done cross validation is 15 and that with sklearn is 17, they are similar in nature. Though the sklearn function is faster!

Similarly the hyper parameters for decision tree are also optimizzed using cross-validation.

```
#Grid search for decision tree
parameters2 = {"criterion": ["gini", "entropy"],
               "min_samples_split": [2, 5, 10, 15, 20, 25, 30, 40, 50],
               "max_depth": [None, 2, 5, 10, 15, 20, 25, 30, 40, 50],
               "min_samples_leaf": [1, 5, 10, 15, 20],
               "max_leaf_nodes": [None, 5, 10, 20, 30, 40, 50],
              }

#Base model
treeNew = DecisionTreeClassifier()

#cv
grid2 = GridSearchCV(treeNew,param_grid = parameters2,cv=7,scoring='accuracy')
grid2.fit(xTrain, yTrain)
resultKnn = pd.DataFrame(grid2.cv_results_)
```

The new accuracies for optimal parameters is an improvement to the previous values got by using random values. On validating the two models on the train data,



Boxplot1 is for kNN and Boxplot2 is for Decision tree; It can be seem that the variance is almost same between the two models, but there is a considerbly small difference in the means of the validation scores for these two methods.

Bonus: Recommender system using similarity measures

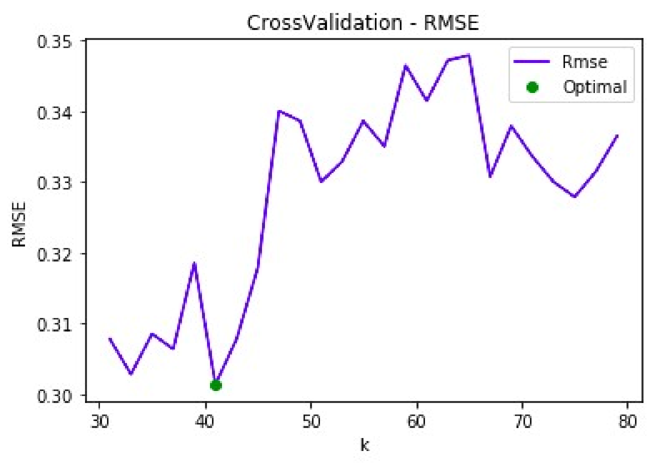
Due to computational deficiencies in the personal laptop, a sub-sample data of movielens 100k dataset is used. This can be one of the major reasons for difference in RMSE values between the reference given for the same.

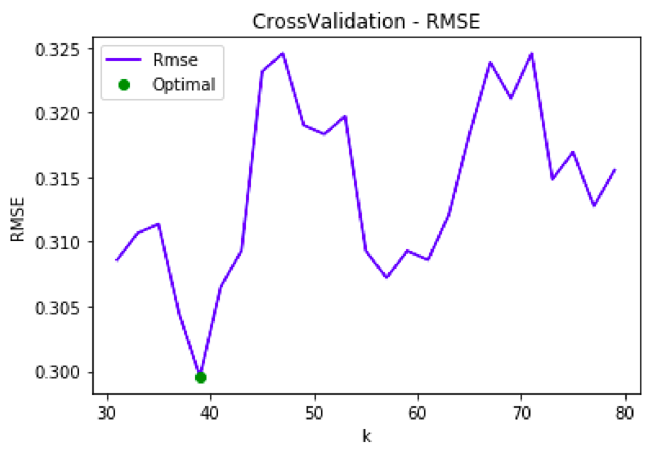
The same kNN function is used, but with linear regression(i.e, mean of y value of k nearest neighbors), and cosine distance is used instead of euclidean distance. A cosine similarity of 1 denotes the vectors are far apart and that of 0 denotes they are close to each other.

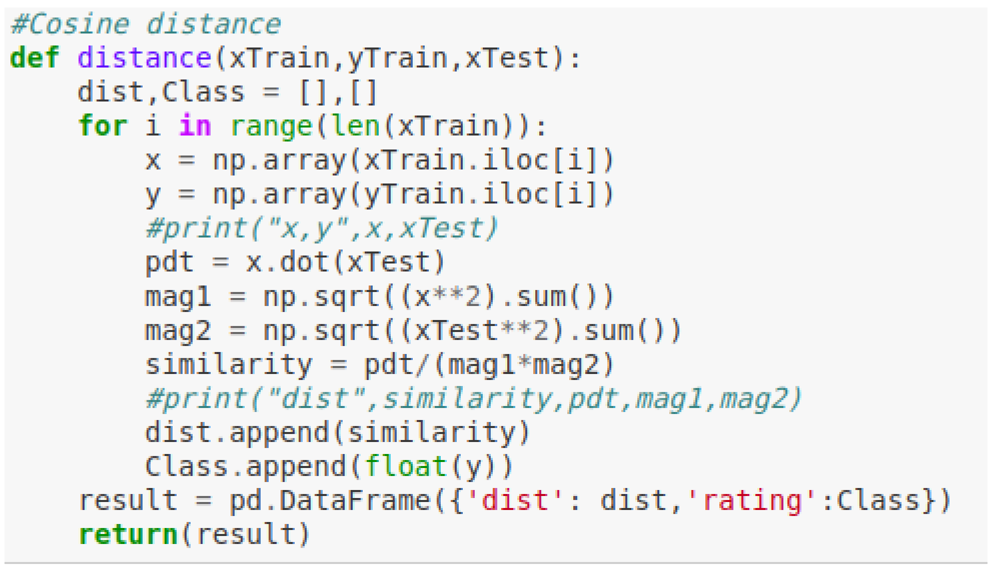
For the item-based kNN prediction, the values are predicted based on the cosine similarity between the parameters that influence the item. While for user-based prediction, the similarity is computed between variables that infuence the user. Or the similarity in items between test and train is used for item-based and user-based prediction depends on how similar the test and train users are!

The data is seperatly joined to item and user to compute the respective predictions. The pre-processing is carried out, where the columns with less information are removed and normalization is done when necessary.

Optimal k value is got by cross- validation and on using them to predict a (70-30) split of sample size of 2000 the RMSE values are `0.052493385826745405` for item-based and `0.049213396483773236` for user-based. The cross validation yeilded a k value of 41 and 39 for the above mentioned two methods.







```
#Cosine distance
def distance(xTrain,yTrain,xTest):
    dist,Class = [],[]
    for i in range(len(xTrain)):
        x = np.array(xTrain.iloc[i])
        y = np.array(yTrain.iloc[i])
        #print("x,y",x,xTest)
        pdt = x.dot(xTest)
        mag1 = np.sqrt((x**2).sum())
        mag2 = np.sqrt((xTest**2).sum())
        similarity = pdt/(mag1*mag2)
        #print("dist",similarity,pdt,mag1,mag2)
        dist.append(similarity)
        Class.append(float(y))
    result = pd.DataFrame({'dist': dist,'rating':Class})
    return(result)
```

Also, for User-based the zipcode contains a few alpha numeric values, considering them as categories will give a huge sparse matrix and they cannot be converted to numerals as well. Considering the fairly small number of their occurence, the rows with them are dropped in pre-processing. Hot-one encoding is done for the rest of the categorical variables.

Comparing results from http://www.mymedialite.net/examples/datasets.html,

	RMSE Computed	MyMediaLite RMSE
Item based	0.0525	0.924
User based	0.0492	0.937

The mymedialite results have used k-value of 40, almost close to what we have computed using cross-validation. There is a difference in the RMSE values as we are considering only a portion of the huge dataset. On scaling up we may end up with higher RMSE values.