KNN & Condensed KNN

UNiversity of north carolina at charlotte | MACHINE LEARNING

Implementition of knn algorithm in python

IMPLEMENTED BY: Krunal Dholakia (800888856) Guided by: Prof RICHARD SOUVEnir

2016

**INDEX**

1. **ALGORITHM – PSUDEO CODE** 
   1. **kNN**
   2. **condensed kNN**
2. **PERFORMANCE MEASUREMENT**
   1. **Performance of teskNN with various value of K and N**
   2. **Performance of condesedkNN with various value of K and N**
3. **CONCLUSION**
4. **APPENDIX**
5. **Algorithm:**

* **K – Nearest Neighbor**

Step 1: Calculate the Euclidean distance between one test point with all training points and store into distance matrix

Step 2: Merge distance and training point’s label into temporary array

Step 3: Sort temporary array with respect to distance in ascending order

Step 4: Take first ‘k’ elements from temporary array as Nearest Neighbors

Step 5: Find most common Nearest Neighbors class as predicted class for that particular test point

**def testkNN**(self,trainX,trainY,testX,k)**:** *#print(datetime.datetime.now())* p**=**[]  
 **for** y **in** range(len(testX))**:** dist**=**[]  
 dist**=**np.linalg.norm(trainX**-**testX[y],axis**=**1)  
 temp**=**zip(dist,trainY)  
 sortedtemp **=** sorted(temp,key**=lambda** tup**:** tup[0])  
 Nearestneighbors**=**sortedtemp[**:**k]  
 NearestneighborsClass **=** map(operator.itemgetter(1), Nearestneighbors)  
 c**=**Counter(NearestneighborsClass).most\_common()  
 p.append(c[0][0])  
 **return** p

* **Condensed 1- Nearest Neighbor**

Step 1: Initialize dummy array with indices of training set, take subsets with first elements and label.

Step 2: Take any other elements from training set

Step 3: apply 1NN algorithm, find predicted class

Step 4: If predicted and actual class is different assign it into subset

Step 5: Repeat process from step 2 till the last element from training set

Step 6: Return set of indices as optimized training set

**def condensedata**(self,trainX, trainY)**:** index**=** []  
 SS**=**[]  
 SSL**=**[]  
 temp**=**[]  
 condensedIdx**=**[]  
 index **=** range(len(trainX))  
 SS.append(trainX[0])  
 SSL.append(trainY[0])  
 **while** sum(index)**:** nonzero\_index **=** np.nonzero(index)  
 RI**=**random.choice(nonzero\_index[0])  
 index[RI]**=**0  
 temp**=**[]  
 temp.append(trainX[RI])  
 predictedtestY **=** self.testkNN(SS,SSL,temp,1)  
 **if**(predictedtestY[0] **!=** trainY[RI])**:** SS.append(trainX[RI])  
 SSL.append(trainY[RI])  
 condensedIdx.append(RI)  
 **return** condensedIdx

**Design decision for implementing C-NN:**

1 – Creation of dummy index to avoid repetition and iterate loop

1. **Performance Measurement :**

* **K – Nearest Neighbor**
* Table of running **time (seconds)** with respect to value of K and N

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K\N | N = 100 | N = 1000 | N = 2000 | N = 5000 | N= 10000 | N = 15000 |
| K=1 | 0.39 | 3.32 | 6.88 | 21.31 | 47.39 | 74.46 |
| K=3 | 0.41 | 3.23 | 7.00 | 21.23 | 45.71 | 74.04 |
| K=5 | 0.41 | 3.20 | 6.74 | 21.28 | 46.12 | 74.97 |
| K=7 | 0.43 | 3.25 | 6.81 | 22.22 | 46.81 | 74.44 |
| K=9 | 0.45 | 3.23 | 6.80 | 20.95 | 45.83 | 75.44 |

* Table of **accuracy** with respect to value of K and N

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K\N | N = 100 | N = 1000 | N = 2000 | N = 5000 | N= 10000 | N = 15000 |
| K=1 | 34.56 | 75.68 | 83.12 | 90.96 | 94.04 | 95.44 |
| K=3 | 26.58 | 69.76 | 79.20 | 89.68 | 93.66 | 95.14 |
| K=5 | 27.4 | 67.70 | 77.50 | 89.06 | 93.12 | 94.50 |
| K=7 | 23.34 | 65.16 | 75.90 | 88.16 | 92.94 | 94.66 |
| K=9 | 20.74 | 62.62 | 74.12 | 87.00 | 92.52 | 94.58 |

* **Condensed Nearest Neighbor**
* Table of running time and **# of rows** in the subset with respect to value of N

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | N=100 | N=1000 | N=2000 | N=5000 | N=10000 | N=15000 |
| # of rows in Subset | 75 | 430 | 662 | 1174 | 1724 | 2282 |

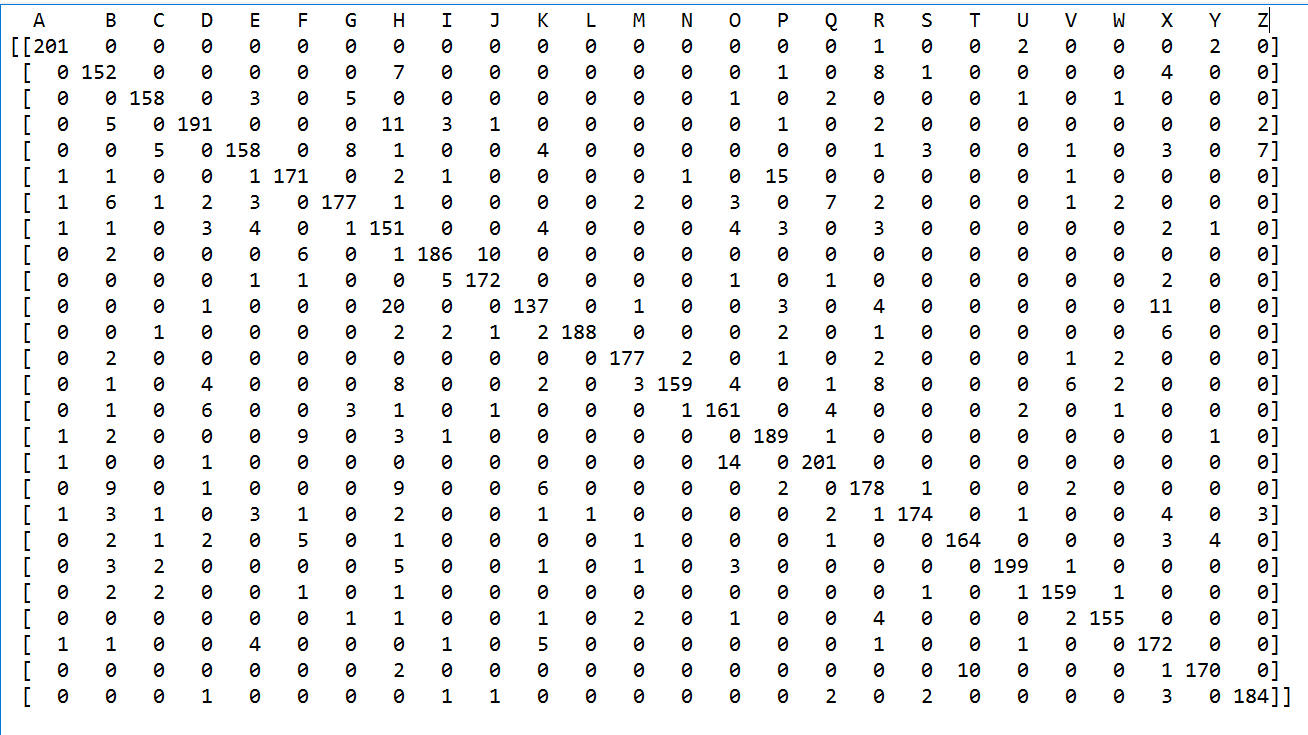
* Table of **running time (seconds)** with respect to value of K and N(running on the condensed subset)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K\N | N = 100 | N = 1000 | N = 2000 | N = 5000 | N= 10000 | N = 15000 |
| K=1 | 0.35 | 1.83 | 3.50 | 9.94 | 26.36 | 49.46 |
| K=3 | 0.36 | 1.81 | 3.37 | 9.71 | 25.43 | 48.24 |
| K=5 | 0.36 | 1.84 | 3.17 | 9.29 | 25.43 | 48.53 |
| K=7 | 0.43 | 1.78 | 3.28 | 9.83 | 25.30 | 48.46 |
| K=9 | 0.39 | 1.86 | 3.27 | 9.41 | 25.20 | 47.95 |

* Table of **accuracy** with respect to value of K and N(running on the condensed subset)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K\N | N = 100 | N = 1000 | N = 2000 | N = 5000 | N= 10000 | N = 15000 |
| K=1 | 33.62 | 70.14 | 78.98 | 85.36 | 89.86 | 91.22 |
| K=3 | 24.56 | 52.6 | 60.06 | 71.18 | 80.08 | 82.32 |
| K=5 | 24.42 | 53.56 | 56.78 | 69.64 | 77.32 | 79.94 |
| K=7 | 19.66 | 48.22 | 54.84 | 66.88 | 73.32 | 78.72 |
| K=9 | 18.46 | 43.02 | 52.3 | 60.86 | 69.12 | 72.4 |

**Confusion Matrix for the value of K = 3 and N = 5000**



1. **Conclusion : Discussion of various experiments**

**Accuracy:**

* + Accuracy of algorithm is decreasing with increasing value of K and decreasing value of N.

**Time:**

* + Time of algorithm is increasing with N.
  + There is not any direct correlation of value of K and TIME from my observation. It is almost same.

**Use of Condensed training set:**

* + Condensing is reducing time in compromise of accuracy.
  + User can opt of condensing if time is more important than accuracy
  + Condensing is working much better for larger training set rather than smaller ones.

1. **Appendix**

Code –

\_\_author\_\_ **=** 'Krunal'  
**import** csv  
**import** random  
**import** math  
**import** operator  
**import** numpy **as** np  
**import** pandas **as** pd  
**from** collections **import** Counter  
**from** sklearn.metrics **import** confusion\_matrix  
**import** random  
**import** datetime  
  
**import** scipy **as** sp  
**from** scipy.spatial **import** distance  
**from** math **import\*  
  
class kNN**(object)**:  
 def testkNN**(self,trainX,trainY,testX,k)**:** *#print(datetime.datetime.now())* p**=**[]  
 **for** y **in** range(len(testX))**:** dist**=**[]  
 dist**=**np.linalg.norm(trainX**-**testX[y],axis**=**1)  
 temp**=**zip(dist,trainY)  
 sortedtemp **=** sorted(temp,key**=lambda** tup**:** tup[0])  
 Nearestneighbors**=**sortedtemp[**:**k]  
 NearestneighborsClass **=** map(operator.itemgetter(1), Nearestneighbors)  
 c**=**Counter(NearestneighborsClass).most\_common()  
 p.append(c[0][0])  
 **return** p  
  
 **def modelaccuracy**(self,testY, predictiontestY)**:** tp **=** 0  
 **for** x **in** range(len(testY))**:  
 if** testY[x][**-**1] **==** predictiontestY[x]**:** tp **+=** 1  
 total **=** float(len(testY))  
 **return** (tp**/**total) **\*** 100.0  
  
 **def condensedata**(self,trainX, trainY)**:** index**=** []  
 SS**=**[]  
 SSL**=**[]  
 temp**=**[]  
 condensedIdx**=**[]  
 index **=** range(len(trainX))  
 SS.append(trainX[0])  
 SSL.append(trainY[0])  
 **while** sum(index)**:** nonzero\_index **=** np.nonzero(index)  
 RI**=**random.choice(nonzero\_index[0])  
 index[RI]**=**0  
 temp**=**[]  
 temp.append(trainX[RI])  
 predictedtestY **=** self.testkNN(SS,SSL,temp,1)  
 **if**(predictedtestY[0] **!=** trainY[RI])**:** SS.append(trainX[RI])  
 SSL.append(trainY[RI])  
 condensedIdx.append(RI)  
 **return** condensedIdx  
  
*#####################################################################*nTrain **=** 5000  
nTest **=** 15000  
k**=**3  
df **=** pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/letter-recognition/letter-recognition.data', header**=**None)  
*#df = pd.read\_csv('C://Users/Krunal/Documents/DSBA/Spring 2016/ML/letter-recognition.data.txt', header=None)*trainX **=** np.array(df.iloc[0**:**nTrain,1**:**])  
trainY **=** np.array(df.iloc[0**:**nTrain,0])  
testX **=** np.array(df.iloc[nTest**:**,1**:**])  
testY **=** np.array(df.iloc[nTest**:**,0])  
condensedtrainX **=** []  
condensedtrainY **=** []  
time1**=**datetime.datetime.now()  
knn **=** kNN()  
**print** 'Train set: ' **+** repr(len(trainX))  
**print** 'Test set: ' **+** repr(len(testX))  
**print** 'K Value :' **+** repr(k)  
*#knn.testkNN1(trainX,trainY,testX,k)*predictedtestY1 **=** knn.testkNN(trainX,trainY,testX,k)  
*#print(predictedtestY1)*accuracy**=**knn.modelaccuracy(testY,predictedtestY1)  
time2**=**datetime.datetime.now()  
t**=**time2**-**time1  
**print**('Time with entire training set:')  
**print**(t)  
**print**('Accuracy with entire training set')  
**print**(accuracy)  
c**=**confusion\_matrix(testY,predictedtestY1,labels**=**np.unique(trainY))  
**print**(c)  
**print**('############################################# ')  
**print** 'Train set: ' **+** repr(len(trainX))  
**print** 'Test set: ' **+** repr(len(testX))  
**print** 'K Value :' **+** repr(k)  
time3**=**datetime.datetime.now()  
condensedIdx**=**knn.condensedata(trainX,trainY)  
**print**(condensedIdx)  
**for** i **in** condensedIdx**:** condensedtrainX.append(trainX[i])  
 condensedtrainY.append(trainY[i])  
  
condensedtrainXtemp **=** np.array(condensedtrainX)  
  
*#print(condensedIdx)***print**(len(condensedIdx))  
predictedtestY **=** knn.testkNN(condensedtrainXtemp,condensedtrainY,testX,k)  
accuracy1**=**knn.modelaccuracy(testY,predictedtestY)  
time4**=**datetime.datetime.now()  
t1**=**time4**-**time3  
**print**('Time with Condensed training set')  
**print**(t1)  
**print**('Accuracy with Condensed training set')  
**print**(accuracy1)