CSC 535 Data Mining

Assignment 2 Report Collection

Submitted to:

Dr. Jamil Saquer

Author(s):

*Chase Dickerson*

**Report**

**Introduction**

The data sets used for this assignment were MNIST\_train.csv and MNIST\_test.csv. They were required to be used for this assignment. Both data sets had headers for each column. Each data set also had a column called “labels” that represented the class to be classified. The classes ranged from zero to nine.

**Background**

The algorithm used for this assignment was the K-nearest neighbors algorithm (KNN). KNN is a distance-based classification algorithm. KNN uses the distance of an item to every other item to determine classification. There are several distance methods to use. I used the Euclidean distance to calculate the distance between points. I also used weighted voting for determining the class. In weighted voting, the vote of a neighbor is inversely proportional to its distance to the test sample.

**Implementation**

I implemented my KNN algorithm based on the pseudocode provided in the lecture slides. I determined my K value by finding the number of classes and taking the square root of that number. Figure 1 demonstrates my implementation of that.

My KNN algorithm includes finding the closest K neighbors and voting on the classes. Weighted voting is used for voting. I did this by taking the inverse of the distance squared. Once voting for a class is finished, the class is returned to the main class. Figure 2 shows my voting implementation.

# Voting for the predicted class using weights

votes **=** **{}**

distanceCount **=** 0

**for** n **in** range**(**len**(**neighbors**)):**

response **=** neighbors**[**n**][**0**]**

**if** response **in** votes**:**

votes**[**response**]** **+=** distances**[**distanceCount**]**

**else:**

votes**[**response**]** **=** distances**[**distanceCount**]**

distanceCount **=** distanceCount **+** 1

predictedClass **=** max**(**votes**.**items**(),**key**=**operator**.**itemgetter**(**1**))[**0**]**

**return** predictedClass

Figure 2: Voting on classes

numberOfClasses **=** **[]**

**for** x **in** range**(**len**(**testData**)):**

c **=** testData**[**x**][**0**]**

**if** c **not** **in** numberOfClasses**:**

numberOfClasses**.**append**(**c**)**

K **=** math**.**floor**(**math**.**sqrt**(**len**(**numberOfClasses**)))**

Figure 1: Finding the value of K

**Experimental Setup and Results**

*Talk about how you setup your experiment. Give any performance results using standard performance metrics here. If you have graphs to illustrate your performance then put them here as well. Show output from the console or from your application here if necessary (as a picture or a table).*

I simply used the data sets provided to carry out the setup of my experiment. I had an 84.00% accuracy rate in classifying classes. I tested 50 test samples and misclassified 8 of the samples. My output is as follows:

K = 3

Desired class: '0', computed class: '0'

Desired class: '0', computed class: '0'

Desired class: '0', computed class: '0'

Desired class: '0', computed class: '0'

Desired class: '0', computed class: '0'

Desired class: '1', computed class: '1'

Desired class: '1', computed class: '1'

Desired class: '1', computed class: '1'

Desired class: '1', computed class: '1'

Desired class: '1', computed class: '1'

Desired class: '2', computed class: '3'

Desired class: '2', computed class: '2'

Desired class: '2', computed class: '2'

Desired class: '2', computed class: '7'

Desired class: '2', computed class: '2'

Desired class: '3', computed class: '7'

Desired class: '3', computed class: '3'

Desired class: '3', computed class: '3'

Desired class: '3', computed class: '3'

Desired class: '3', computed class: '3'

Desired class: '4', computed class: '4'

Desired class: '4', computed class: '4'

Desired class: '4', computed class: '4'

Desired class: '4', computed class: '4'

Desired class: '4', computed class: '9'

Desired class: '5', computed class: '5'

Desired class: '5', computed class: '6'

Desired class: '5', computed class: '5'

Desired class: '5', computed class: '5'

Desired class: '5', computed class: '5'

Desired class: '6', computed class: '0'

Desired class: '6', computed class: '6'

Desired class: '6', computed class: '6'

Desired class: '6', computed class: '6'

Desired class: '6', computed class: '6'

Desired class: '7', computed class: '7'

Desired class: '7', computed class: '7'

Desired class: '7', computed class: '7'

Desired class: '7', computed class: '7'

Desired class: '7', computed class: '7'

Desired class: '8', computed class: '8'

Desired class: '8', computed class: '8'

Desired class: '8', computed class: '8'

Desired class: '8', computed class: '3'

Desired class: '8', computed class: '8'

Desired class: '9', computed class: '9'

Desired class: '9', computed class: '7'

Desired class: '9', computed class: '9'

Desired class: '9', computed class: '9'

Desired class: '9', computed class: '9'

Accuracy rate: 84.0

Number of misclassified test samples: 8

Total number of test samples: 50

**Conclusion**

In conclusion, I learned more about classification. I learned a lot about the classification algorithm KNN. I had good results and good accuracy using the test data provided.

**Code**

#!/usr/bin/env python3

# Program: KNN algorithm

# Developer: Chase Dickerson

# Date: 10/07/2019

**import** sys

**import** csv

**import** math

**import** operator

#Global Variables

trainingData **=** **[]**

testData **=** **[]**

# Reads the training and test data in

**def** preprocessData**(**trainingFile**,** testFile**):**

**with** open**(**trainingFile**,** 'r'**)** **as** csvFile1**:**

trainingLines **=** csv**.**reader**(**csvFile1**)**

dataSet1 **=** list**(**trainingLines**)**

**global** trainingData

**for** x **in** range**(**len**(**dataSet1**)):**

trainingData**.**append**(**dataSet1**[**x**])**

**with** open**(**testFile**,** 'r'**)** **as** csvFile**:**

testLines **=** csv**.**reader**(**csvFile**)**

dataSet2 **=** list**(**testLines**)**

**global** testData

**for** x **in** range**(**len**(**dataSet2**)):**

testData**.**append**(**dataSet2**[**x**])**

# Ecludian distance is used to find the distance between 2 points

**def** ecludianDistance**(**p**,** q**,** l**):**

distSum **=** 0

**for** x **in** range**(**l**):**

distSum **+=** pow**((**int**(**p**[**x **+** 1**])** **-** int**(**q**[**x **+** 1**])),** 2**)**

**return** math**.**sqrt**(**distSum**)**

# KNN algorithm used to predict a class

**def** kkn**(**trainingData**,** K**,** test**):**

neighbors **=** **[]**

# Finds K closest neighbors

**for** d **in** range**(**len**(**trainingData**)):**

**if** len**(**neighbors**)** **<=** K**:**

neighbors**.**append**(**trainingData**[**d**])**

**else:**

**for** n **in** neighbors**:**

**if** ecludianDistance**(**test**,** n**,** len**(**test**)** **-** 1**)** **>=** ecludianDistance**(**test**,** trainingData**[**d**],** len**(**test**)** **-** 1**):**

neighbors**.**remove**(**n**)**

neighbors**.**append**(**trainingData**[**d**])**

# Weighted voting

distances **=** **[]**

**for** n **in** neighbors**:**

distance **=** 1 **/** pow**(**ecludianDistance**(**test**,** n**,** len**(**test**)** **-** 1**),** 2**)**

distances**.**append**(**distance**)**

# Voting for the predicted class using weights

votes **=** **{}**

distanceCount **=** 0

**for** n **in** range**(**len**(**neighbors**)):**

response **=** neighbors**[**n**][**0**]**

**if** response **in** votes**:**

votes**[**response**]** **+=** distances**[**distanceCount**]**

**else:**

votes**[**response**]** **=** distances**[**distanceCount**]**

distanceCount **=** distanceCount **+** 1

predictedClass **=** max**(**votes**.**items**(),** key**=**operator**.**itemgetter**(**1**))[**0**]**

**return** predictedClass

# Prints data on the accuracy

**def** getAccuracyData**(**computedClasses**,** testData**):**

correct **=** 0

incorrect **=** 0

**for** x **in** range**(**len**(**testData**)):**

**if** testData**[**x**][**0**]** **==** computedClasses**[**x**]:**

correct **+=** 1

**else:**

incorrect **+=** 1

accuracyRate **=** **(**correct**/**float**(**len**(**testData**)))** **\*** 100.0

**print(**'Accuracy rate: ' **+** repr**(**accuracyRate**))**

**print(**'Number of misclassified test samples: ' **+** repr**(**incorrect**))**

**print(**'Total number of test samples: ' **+** repr**(**len**(**testData**)))**

**def** main**():**

preprocessData**(**'MNIST\_train.csv'**,** 'MNIST\_test.csv'**)**

trainingData**.**pop**(**0**)** #Removing headers

testData**.**pop**(**0**)** #Removing headers

predictedClasses **=** **[]**

numberOfClasses **=** **[]**

**for** x **in** range**(**len**(**testData**)):**

c **=** testData**[**x**][**0**]**

**if** c **not** **in** numberOfClasses**:**

numberOfClasses**.**append**(**c**)**

K **=** math**.**floor**(**math**.**sqrt**(**len**(**numberOfClasses**)))**

**print(**"K = " **+** repr**(**K**))**

**for** x **in** range**(**len**(**testData**)):**

klass **=** kkn**(**trainingData**,** K**,** testData**[**x**])**

predictedClasses**.**append**(**klass**)**

**print(**'Desired class: ' **+** repr**(**testData**[**x**][**0**])** **+** ', computed class: ' **+** repr**(**klass**))**

getAccuracyData**(**predictedClasses**,** testData**)**

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

main**()**