CSC 535 Data Mining

## Assignment 3 Report

### Submitted to:

### Dr. Jamil Saquer

### Author(s):

### Khaled Hossain

**K-nearest neighbor**

**Introduction**

For this assignment, the task was to implement the weighted voting for k-nearest neighbor algorithm. I used the dataset provided by the instructor. The data set is named digit recognizer. The data files MNIST\_train.csv and MNIST\_test.csv contain gray-scale images of hand-drawn digits, from zero through nine. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. The training data set had 785 columns. The first column is called "label" and it is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

**Background**

In this assignment I used weighted voting in my implementation and the Euclidean distance to find the k-nearest neighbors. **In weighted voting**, the vote of a neighbor is inversely proportional to its distance to the test sample. I also tried to use the **Cosine similarity** which is a measure of similarity between two non-zero vectors of an inner product space. Lastly, I also tried to normalize the data and to check if it would improve the accuracy of the KNN algorithm. There are many ways to calculate the distance between two neighbors, such as the Manhattan distance, cosine distance, Minkowski distance. We normalize the data to bring all the variables to the same range.

**Implementation**

After preparing the data, I started to implement the functions the functions. When choosing the optimal of k I experimented with numerous values, however the accuracy was the highest when I used k = 4. And as I increased the value of K the accuracy decreased.

**Experimental Setup and Results**

After checking the program with numerous k values. I decided the optimal k value for the experiment to be 4. This gave us an accuracy of 86% with only 7 misclassified classes. I tried to implement the cosine similarity and to scale and normalize the data but my implementation was incorrect as it gave me poorer result for accuracy.

**Conclusion**

I tried multiple ways to improve the accuracy. Firstly, I tried to normalize the data. The k-nearest neighbor algorithm relies on majority voting. Without normalization, all the nearest neighbors are aligned in the direction of the axis with the smaller range. But if we normalize the data this is fixed. However, I was not able to implement it properly because when I tried running my code my accuracy decreased. Secondly, min max scaling of the data also improves the performance of KNN algorithm. The performance of KNN requires preprocessing of data to make all variables similarly scaled and centered. But I wasn’t able to implement this correctly.

**References**

*Digit Recognizer. (n.d.).*

*Retrieved October 23, 2020, from https://www.kaggle.com/c/digit-recognizer/data*

**Code**

|  |
| --- |
| knn.py |

1 *"""*

2 *Program: hw3.ipynb*

3 *Programmed by: Khaled Hossain*

4 *Description: An implementation of KNN algorithm*

5 *"""*

6 *# ---------------------------------Imports--------------------------------------*

7 **import** numpy **as** np

8 **import** pandas **as** pd

9 **from** sklearn **import** preprocessing

10 **from** operator **import** itemgetter

11

12 *# ------------------------------------------------------------------------------*

13

14 *# Get the data*

15 train\_data = pd.read\_csv(**'MNIST\_train.csv'**)

16 test\_data = pd.read\_csv(**'MNIST\_test.csv'**)

17 X\_train = np.array(train\_data)

18 X\_test = np.array(test\_data)

19

20

21 *# Scale & normalize (gave worse accuracy)*

22 **def** normalize(data):

23 **return** preprocessing.normalize(data)

24

25

26 **def** scale(data):

27 **return** preprocessing.scale(data)

28

29

30 *# Calculates the distance between samples*

31 **def** euclidean\_distance(testData, trainData):

32 distance = 0

33

34 **for** i **in** range(len(testData)):

35 distance += (testData[i] - trainData[i]) \*\* 2

36

37 **return** np.sqrt(distance)

38

39

40 *# cosine\_similarity (gave worse accuracy)*

41 **def** cos\_sim(a, b):

42 dot\_product = np.dot(a, b)

43 norm\_a = np.linalg.norm(a)

44 norm\_b = np.linalg.norm(b)

45

46 **return** dot\_product / (norm\_a \* norm\_b)

47

48

49 *# ---------------------------------Classes/Functions----------------------------*

50

51 **class** KNN:

52 **def** \_\_init\_\_(self, k=4):

53 self.k = k

54

55 *# Calculate Accuracy*

56 **def** accuracy(self, testData, outputData):

57 match = 0

58 mistake = 0

59 **for** i **in** range(len(testData)):

60 **if** testData[i][0] == outputData[i]:

61 match += 1

62 **else**:

63 mistake += 1

64

65 accuracy = (match / len(outputData)) \* 100

66 **return** [accuracy, mistake]

67

68 *# get the weights for closest neighbors*

69 *# compute the inverse of each distance, find the sum of the inverses, then divide each inverse by the sum*

70 **def** get\_weights(self, data):

71

72 **for** i **in** data:

73 weight = 1 / i[1]

74 i.append(weight)

75

76 *# print(data)*

77 data.sort(key=itemgetter(2))

78 *# print(data)*

79 *# now we have [Class, distance, weight]*

80 **return** data[0][0]

81

82 *# weighted K-nearest neighbor algorthim*

83 **def** weighted\_knn(self, testData, trainData):

84

85 outputData = []

86

87 print(**'K = '**, self.k)

88

89 **for** data **in** testData:

90 distance = []

91

92 *# using euclidean distance*

93 **for** t\_data **in** trainData:

94 distance.append([t\_data[0], euclidean\_distance(data[1:], t\_data[1:])])

95 distance.sort(key=itemgetter(1))

96

97 *# using cosine similarity*

98 *# for train in trainData:*

99 *# distance.append([train[0], cos\_sim(data[1:], train[1:])])*

100 *# distance.sort(key=itemgetter(1))*

101

102 neighbors = []

103

104 **for** i **in** range(self.k):

105 neighbors.append(distance[i])

106

107 result = self.get\_weights(neighbors)

108 outputData.append(result)

109 *# print(outputData)*

110

111 print(**'Desired Class = '**, data[0], **"Computed Class = "**, result)

112

113 *# Display accuracy, misclassifications and total number of test samples*

114 accuracy\_result = self.accuracy(testData, outputData)

115 print(**'Accuracy Rate = '**, accuracy\_result[0], **"%"**)

116 print(**'Number of misclassified test samples = '**, accuracy\_result[1])

117 print(**'Total number of test samples = '**, len(testData))

118

119

120 *# calling the classifier and setting the value of k to 4*

121

122 *# noramlized data*

123 *# a = normalize(X\_train)*

124 *# b = normalize(X\_test)*

125 *# clf = KNN(k = 4)*

126 *# clf.weighted\_knn(b, a)*

127

128

129 clf = KNN(k=4)

130

131 clf.weighted\_knn(X\_test, X\_train)

132

133

134