Project 2 CS170: Introduction to Artificial Intelligence

Osvaldo Moreno Dr. Eamonn Keogh

862016789

omore005@ucr.edu

12-12-2019

In Completing this assignment, I consulted:

* Lecture Slides for the Nearest Neighbor Search provided by Dr. Keogh
* T.A Ryan to help me understand better, refactor code, and debug some issues
* Several Python documentation pages:
  + [https://devdocs.io/python](https://devdocs.io/python/)/
  + <https://numpy.org/doc/1.17/>
* Test Data was provided by the Dr. Keogh
* Dr. Keogh Helped understand the algorithm better and debug some issues
* Theory came from class text book
* Report format inspired from example provided by Dr. Keogh

All Important code is original. Unimportant routines that are not completely original are….

* The user interface was inspired from the sample report with permission of Dr. Keogh.
* Read and use data from this video: <https://www.youtube.com/watch?v=dR1JZjM8WYU>
* Routines to calculate distances where made by me

CS170: Project 1 Write Up

Osvaldo Moreno Ornelas, SID 862016789

**Introduction**:

For this project we will use the Nearest Neighbor algorithm in order to find the best set of features in a dataset. We will test several datasets in different versions of the algorithm.

**Nearest Neighbor Algorithm**

For this algorithm we locate the neighbors of the points that we best think are a fit for our model. Perhaps we see if there exists a better feature based on its Euclidean distance. We do not go through all the features, but instead through a subset of only K amount of features, thus making it the K-Nearest Neighbor algorithm. We only add features that give us back the best accuracy which is correct distance divided by the size of the instances.

**Approaches Taken**

For this project we took 3 different algorithm approaches: Forward Search, Backwards elimination search, and a custom one.

**Forward search:** starts from empty and adds only the features with the best accuracy. Then it compares it starts again but only comparing if not already in the set of best features (to avoid duplicate checks).

**Backwards elimination**: For this one we start the feature array has all the features (1-n). As the algorithm is run, it eliminates the worst features. At the end only the best ones are left. Instead of running the search from 1 to n, we run it from n to 1.

**Custom Algorithm**: For this algorithm, I grabbed forward selection and added a feature that halts the search if the search fails at least a 3rd of the cases. The goal is to only test if it is worth it to keep going forward rather than keep wasting time on cases that will not work.

**Test Data**

We will show the best features outputted and the best accuracy. The datasets used are small set 120, small set 82, and large set 84. Large set was only tested in forward selection due to the amount of time it takes to test the data.

**Results From Each Algorithm**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Best Features |  |
|  |  |  |  |
|  | Small 120 | Small 82 | Large 84 |
| Forward | [10, 7] | [1, 4] | [51, 66, 80] |
| Backwards | [7, 10] | [1, 4] |  |
| Custom | [1, 5, 7, 4] | [1, 4] |  |
|  |  |  |  |
|  |  |  |  |
|  |  | Best Features |  |
|  |  |  |  |
|  | Small 120 | Small 82 | Large 84 |
| Forward | 0 .95 | 0.97 | 0.95 |
| Backwards | 0.95 | 0.97 |  |
| Custom | 0.77 | 0.97 |  |
|  |  |  |  |

The accuracy and the set of features is consistent with most of the data sets. The only issue comes from the custom set. When it comes to regular algorithms, there is no difference in between the sets and both are successful and worked as necessary.

**Custom Algorithm Results**

The results stayed constant for small test data 82. Because of this I performed a speed test on that data set. The custom algorithm was compared against forward search. After timing both algorithms the results are as followed:

Forward Search: 45.1 seconds

Custom Search: 35.9 seconds

Difference: 9.2 seconds

Time improvement: ((45.1 - 35.9) / 45.1 ) \* 100 = 20.4%

On a perfect run, the algorithm is 20.4% faster. This is an improvement from the original test, but it is not reliable. We can see that in the small data set 120, the results was not appropriate. Because of the elimination of unnecessary cases, there is a possibility that the best feature is reached in the features after the elimination, therefore it is not 100% reliable. The algorithm developed is not optimal, but for some sets of data it has the possibility of working.

**Accuracy Graph**

Using small data set 82 with forward search:

|  |  |
| --- | --- |
| Features | Accuracy % |
| 0 | 83.0 |
| 1 | 86.0 |
| 2 | 97.0 |
| 3 | 95.0 |
| 4 | 93.0 |
| 5 | 90.0 |
| 6 | 87.0 |
| 7 | 82.0 |
| 8 | 81.0 |
| 9 | 78.0 |
| 10 | 75.0 |
|  |  |

Default Rate (for empty set) = = \* 100 = 83

**Program Trace**

**Small Data Set 82**:

Welcome to my program. We will run the test using all 3 algorithms

First, Using Forward Search:

The dataset has 10 features, with 300 instances.

On the 1 th level of the search tree

Using Feature(s) [1] accuracy is: 0.86

Using Feature(s) [2] accuracy is: 0.72

Using Feature(s) [3] accuracy is: 0.73

Using Feature(s) [4] accuracy is: 0.74

Using Feature(s) [5] accuracy is: 0.72

Using Feature(s) [6] accuracy is: 0.70

Using Feature(s) [7] accuracy is: 0.70

Using Feature(s) [8] accuracy is: 0.69

Using Feature(s) [9] accuracy is: 0.75

Using Feature(s) [10] accuracy is: 0.72

On the 2 th level of the search tree

Using Feature(s) [1, 2] accuracy is: 0.85

Using Feature(s) [1, 3] accuracy is: 0.84

Using Feature(s) [1, 4] accuracy is: 0.97

Using Feature(s) [1, 5] accuracy is: 0.87

Using Feature(s) [1, 6] accuracy is: 0.84

Using Feature(s) [1, 7] accuracy is: 0.87

Using Feature(s) [1, 8] accuracy is: 0.89

Using Feature(s) [1, 9] accuracy is: 0.81

Using Feature(s) [1, 10] accuracy is: 0.84

…..

On the 9 th level of the search tree

Using Feature(s) [1, 4, 7, 8, 6, 2, 5, 9, 3] accuracy is: 0.74

Using Feature(s) [1, 4, 7, 8, 6, 2, 5, 9, 10] accuracy is: 0.78

On the 10 th level of the search tree

Using Feature(s) [1, 4, 7, 8, 6, 2, 5, 9, 10, 3] accuracy is: 0.75

Set Of Features[1, 4]

Best Accuracy: 0.97

Second, Using Backward Search:

The dataset has 10 features, with 300 instances.

On the 1 th level of the search tree

Using Feature(s) [2, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is: 0.70

Using Feature(s) [1, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is: 0.79

Using Feature(s) [1, 2, 4, 5, 6, 7, 8, 9, 10] accuracy is: 0.78

Using Feature(s) [1, 2, 3, 5, 6, 7, 8, 9, 10] accuracy is: 0.75

Using Feature(s) [1, 2, 3, 4, 6, 7, 8, 9, 10] accuracy is: 0.77

Using Feature(s) [1, 2, 3, 4, 5, 7, 8, 9, 10] accuracy is: 0.76

Using Feature(s) [1, 2, 3, 4, 5, 6, 8, 9, 10] accuracy is: 0.74

Using Feature(s) [1, 2, 3, 4, 5, 6, 7, 9, 10] accuracy is: 0.78

Using Feature(s) [1, 2, 3, 4, 5, 6, 7, 8, 10] accuracy is: 0.74

Using Feature(s) [1, 2, 3, 4, 5, 6, 7, 8, 9] accuracy is: 0.74

On the 2 th level of the search tree

Using Feature(s) [3, 4, 5, 6, 7, 8, 9, 10] accuracy is: 0.70

Using Feature(s) [1, 4, 5, 6, 7, 8, 9, 10] accuracy is: 0.79

Using Feature(s) [1, 3, 5, 6, 7, 8, 9, 10] accuracy is: 0.74

Using Feature(s) [1, 3, 4, 6, 7, 8, 9, 10] accuracy is: 0.78

Using Feature(s) [1, 3, 4, 5, 7, 8, 9, 10] accuracy is: 0.80

Using Feature(s) [1, 3, 4, 5, 6, 8, 9, 10] accuracy is: 0.74

Using Feature(s) [1, 3, 4, 5, 6, 7, 9, 10] accuracy is: 0.77

Using Feature(s) [1, 3, 4, 5, 6, 7, 8, 10] accuracy is: 0.78

Using Feature(s) [1, 3, 4, 5, 6, 7, 8, 9] accuracy is: 0.79

…….

On the 9 th level of the search tree

Using Feature(s) [4] accuracy is: 0.74

Using Feature(s) [1] accuracy is: 0.86

On the 10 th level of the search tree

Using Feature(s) [] accuracy is: 0.16

Set Of Features[1, 4]

Best Accuracy: 0.97

At Last, Using Naughty Search:

The dataset has 10 features, with 300 instances.

On the 1 th level of the search tree

Using Feature(s) [1] accuracy is: 0.86

Using Feature(s) [2] accuracy is: 0.72

Using Feature(s) [3] accuracy is: 0.73

Using Feature(s) [4] accuracy is: 0.74

On the 2 th level of the search tree

Using Feature(s) [1, 2] accuracy is: 0.85

Using Feature(s) [1, 3] accuracy is: 0.84

Using Feature(s) [1, 4] accuracy is: 0.97

Using Feature(s) [1, 5] accuracy is: 0.87

Using Feature(s) [1, 6] accuracy is: 0.84

…….

On the 10 th level of the search tree

Using Feature(s) [1, 4, 2] accuracy is: 0.91

Using Feature(s) [1, 4, 3] accuracy is: 0.91

Using Feature(s) [1, 4, 5] accuracy is: 0.92

Using Feature(s) [1, 4, 6] accuracy is: 0.91

Using Feature(s) [1, 4, 7] accuracy is: 0.95

Using Feature(s) [1, 4, 8] accuracy is: 0.93

Set Of Features[1, 4]

Best Accuracy: 0.97

**Large Data Set 84:**

The dataset has 100 features, with 300 instances.

On the 1 th level of the search tree

Using Feature(s) [1] accuracy is: 0.78

Using Feature(s) [2] accuracy is: 0.77

Using Feature(s) [3] accuracy is: 0.76

Using Feature(s) [4] accuracy is: 0.74

Using Feature(s) [5] accuracy is: 0.76

Using Feature(s) [6] accuracy is: 0.77

Using Feature(s) [7] accuracy is: 0.75

Using Feature(s) [8] accuracy is: 0.77

Using Feature(s) [9] accuracy is: 0.77

Using Feature(s) [10] accuracy is: 0.78

Using Feature(s) [11] accuracy is: 0.77

Using Feature(s) [12] accuracy is: 0.73

Using Feature(s) [13] accuracy is: 0.76

Using Feature(s) [14] accuracy is: 0.75

Using Feature(s) [15] accuracy is: 0.77

Using Feature(s) [16] accuracy is: 0.77

Using Feature(s) [17] accuracy is: 0.74

Using Feature(s) [18] accuracy is: 0.81

Using Feature(s) [19] accuracy is: 0.77

Using Feature(s) [20] accuracy is: 0.77

…….

Using Feature(s) [96] accuracy is: 0.73

Using Feature(s) [97] accuracy is: 0.77

Using Feature(s) [98] accuracy is: 0.75

Using Feature(s) [99] accuracy is: 0.76

Using Feature(s) [100] accuracy is: 0.79

On the 2 th level of the search tree

Using Feature(s) [51, 1] accuracy is: 0.84

Using Feature(s) [51, 2] accuracy is: 0.85

Using Feature(s) [51, 3] accuracy is: 0.86

Using Feature(s) [51, 4] accuracy is: 0.88

Using Feature(s) [51, 5] accuracy is: 0.84

Using Feature(s) [51, 6] accuracy is: 0.83

Using Feature(s) [51, 7] accuracy is: 0.86

Using Feature(s) [51, 8] accuracy is: 0.87

Using Feature(s) [51, 9] accuracy is: 0.85

Using Feature(s) [51, 10] accuracy is: 0.87

……….

Using Feature(s) [51, 93] accuracy is: 0.84

Using Feature(s) [51, 94] accuracy is: 0.88

Using Feature(s) [51, 95] accuracy is: 0.80

Using Feature(s) [51, 96] accuracy is: 0.83

Using Feature(s) [51, 97] accuracy is: 0.84

Using Feature(s) [51, 98] accuracy is: 0.86

Using Feature(s) [51, 99] accuracy is: 0.86

Using Feature(s) [51, 100] accuracy is: 0.85

……..

On the 99 th level of the search tree

Using Feature(s) [51, 66, 80, 70, 29, 6, 24, 48, 98, 69, 26, 1, 85, 46, 10, 37, 33, 88, 82, 11, 63, 60, 28, 13, 32, 59, 15, 3, 55, 67, 25, 17, 100, 61, 38, 44, 42, 45, 35, 22, 40, 72, 5, 78, 76, 8, 91, 94, 30, 90, 14, 7, 16, 43, 68, 77, 75, 20, 41, 39, 56, 12, 27, 87, 53, 84, 54, 89, 31, 52, 64, 93, 21, 57, 4, 86, 49, 92, 65, 96, 81, 97, 58, 34, 79, 95, 99, 73, 19, 36, 74, 18, 62, 71, 47, 23, 9, 2, 50] accuracy is: 0.71

Using Feature(s) [51, 66, 80, 70, 29, 6, 24, 48, 98, 69, 26, 1, 85, 46, 10, 37, 33, 88, 82, 11, 63, 60, 28, 13, 32, 59, 15, 3, 55, 67, 25, 17, 100, 61, 38, 44, 42, 45, 35, 22, 40, 72, 5, 78, 76, 8, 91, 94, 30, 90, 14, 7, 16, 43, 68, 77, 75, 20, 41, 39, 56, 12, 27, 87, 53, 84, 54, 89, 31, 52, 64, 93, 21, 57, 4, 86, 49, 92, 65, 96, 81, 97, 58, 34, 79, 95, 99, 73, 19, 36, 74, 18, 62, 71, 47, 23, 9, 2, 83] accuracy is: 0.72

On the 100 th level of the search tree

Using Feature(s) [51, 66, 80, 70, 29, 6, 24, 48, 98, 69, 26, 1, 85, 46, 10, 37, 33, 88, 82, 11, 63, 60, 28, 13, 32, 59, 15, 3, 55, 67, 25, 17, 100, 61, 38, 44, 42, 45, 35, 22, 40, 72, 5, 78, 76, 8, 91, 94, 30, 90, 14, 7, 16, 43, 68, 77, 75, 20, 41, 39, 56, 12, 27, 87, 53, 84, 54, 89, 31, 52, 64, 93, 21, 57, 4, 86, 49, 92, 65, 96, 81, 97, 58, 34, 79, 95, 99, 73, 19, 36, 74, 18, 62, 71, 47, 23, 9, 2, 83, 50] accuracy is: 0.70

Set Of Features[51, 66, 80]

Best Accuracy: 0.95

**Source Code**

import numpy as np

import math

import copy

def ReadData():

dataList = np.loadtxt("smallTest82.txt")

#dataList = np.loadtxt("largeTest84.txt")

print("The dataset has " + str(np.size(dataList,1)-1) + " features, with "+ str(np.size(dataList,0)) + " instances.\n")

return dataList

def calcDistance(analyzedFeatures, Data, i,j):

distance = 0

for k in range(len(analyzedFeatures)):

distance += pow(Data[i][analyzedFeatures[k]] - Data[j][analyzedFeatures[k]], 2)

return math.sqrt(distance)

def leaveOneOutCrossValidation(analyzeFeatures, setOfFeatures):

correctCount = 0

for i in range(0, np.size(analyzeFeatures,0)): #from 0 to end of rows

currentBest = float("inf")

currentBestIndex = None

#print("I am looping over the rows " + str(i))

for j in range(0, np.size(analyzeFeatures,0)): #from 0 to end of rows

if i != j:

"""

distance = math.sqrt(pow(analyzeFeatures[i][1] - analyzeFeatures[j][1], 2)

+ pow(analyzeFeatures[i][2] - analyzeFeatures[j][2], 2))

"""

distance = calcDistance(setOfFeatures, analyzeFeatures, i,j)

#print("Distance: " + str(distance))

if (distance < currentBest):

currentBest = distance

currentBestIndex = j

if(analyzeFeatures[i][0] == analyzeFeatures[currentBestIndex][0]):

correctCount += 1

accuracy = correctCount / np.size(analyzeFeatures,0)

print("Using Feature(s) " + str(setOfFeatures) + " accuracy is: " + format(accuracy, '.2f'))

return accuracy

"""

tableData is the multi-D array with the data

"""

def searchFunction(tableData):

setOfFeatures = [] #empty list of features

globalFeatures = []

globalAccuracy = 0

for i in range(1, np.size(tableData,1)): #from start to finish of columns

print("\nOn the " + str(i) + " th level of the search tree\n")

featureToAdd = None #best features list

bestAccuracy = 0 #accuracy number

for k in range(1, np.size(tableData,1)):#from start to finish of columns

if k not in setOfFeatures: #if k is not in the features

featuresToTest = copy.deepcopy(setOfFeatures)

featuresToTest.append(k)

accuracy = leaveOneOutCrossValidation(tableData, featuresToTest) #validate feature and get accuracy

if (accuracy > bestAccuracy): #if accuracy is better

bestAccuracy = accuracy #update accuracy

featureToAdd = k#next feature to add because accuracy is better

setOfFeatures.append(featureToAdd) # add to the set of features

#if(featureToAdd not in setOfFeatures):

if(bestAccuracy > globalAccuracy):

globalAccuracy = bestAccuracy

globalFeatures = copy.deepcopy(setOfFeatures)

print("\nSet Of Features"+str(globalFeatures))

print("Best Accuracy: " + format(globalAccuracy, '.2f') + "\n\n")

"""

This function is for backtrace

"""

def searchFunctionBackTrack(tableData):

setOfFeatures = list(range(1, np.size(tableData, 1))) # empty list of features

globalFeatures = []

globalAccuracy = 0

for i in range(1, np.size(tableData,1)): # from start to finish of columns

print("\nOn the " + str(i) + " th level of the search tree\n")

featureToDelete = None # best features list

bestAccuracy = 0 # accuracy number

for k in range(1, np.size(tableData,1)): # from start to finish of columns

if k in setOfFeatures: # if k is not in the features

featuresToTest = copy.deepcopy(setOfFeatures)

featuresToTest.remove(k)

accuracy = leaveOneOutCrossValidation(tableData, featuresToTest) # validate feature and get accuracy

if (accuracy > bestAccuracy): # if accuracy is better

bestAccuracy = accuracy # update accuracy

featureToDelete = k # next feature to add because accuracy is better

setOfFeatures.remove(featureToDelete) # add to the set of features

# if(featureToAdd not in setOfFeatures):

if (bestAccuracy > globalAccuracy):

globalAccuracy = bestAccuracy

globalFeatures = copy.deepcopy(setOfFeatures)

print("\nSet Of Features" + str(globalFeatures))

print("Best Accuracy: " + format(globalAccuracy, '.2f') + "\n\n")

"""

This function is the personl one

"""

def searchFunctionPersonal(tableData):

setOfFeatures = [] # empty list of features

globalFeatures = []

globalAccuracy = 0

for i in range(1, np.size(tableData, 1)): # from start to finish of columns

print("\nOn the " + str(i) + " th level of the search tree\n")

featureToAdd = None # best features list

bestAccuracy = 0 # accuracy number

reduceCount = 0

for k in range(1, np.size(tableData, 1)): # from start to finish of columns

if k not in setOfFeatures: # if k is not in the features

featuresToTest = copy.deepcopy(setOfFeatures)

featuresToTest.append(k)

accuracy = leaveOneOutCrossValidation(tableData, featuresToTest) # validate feature and get accuracy

if (accuracy > bestAccuracy): # if accuracy is better

bestAccuracy = accuracy # update accuracy

featureToAdd = k # next feature to add because accuracy is better

else:

reduceCount += 1

if reduceCount >= math.floor((np.size(tableData, 1)-1) / 3):

break #if the fail count is less than half the amount of features, then break out of the search and move on

if(featureToAdd not in setOfFeatures and bestAccuracy>globalAccuracy):

setOfFeatures.append(featureToAdd) # add to the set of features

# if(featureToAdd not in setOfFeatures):

if (bestAccuracy > globalAccuracy):

globalAccuracy = bestAccuracy

globalFeatures = copy.deepcopy(setOfFeatures)

print("\nSet Of Features" + str(globalFeatures))

print("Best Accuracy: " + format(globalAccuracy, '.2f') + "\n\n")

"""

MAIN PROGRAM

This is the controller for the program, there is no interface, just a run.

"""

print("\n Welcome to my program. We will run the test using all 3 algorithms\n")

print("First, Using Forward Search: \n")

searchFunction(ReadData())

print("Second, Using Backward Search: \n")

searchFunctionBackTrack(ReadData())

print("At Last, Using Naughty Search: \n")

searchFunctionPersonal(ReadData())