Project 2 for CS 205 Spring 2021, with Dr Eamonn Keogh.

Feature Selection with Nearest Neighbor

Prince Choudhary
SID: 862254827
Email: pchou014@cs.ucr.edu

In completing the project, I consulted:

1) Lecture slides.
2) https://www.w3schools.com/python/
3) https://colab.research.google.com/

All code is original, except:

1) I used python libraries like math, operator, re to ease mathematical calculations.
2) I used matplotlib to plot the bar charts.

Introduction:

We are provided with documents with dataset having a number of features. Our aim is to find out the relevant features. We are provided with two datasets small and large. The small data set contains 10 features and large dataset contains 250 features. The datasets consist of with class labels '1' and '2'. We will normalize the data so that they lie between 0 and 1.

Nearest neighbour classifier is used to determine the class label. We are using 'Forward Selection' and 'Backward Elimination' search algorithms. We find the relevant features by the accuracy of nearest neighbour classifier.

I used Python3 to implement this project, I used python libraries like math, operator, re, decimal for mathematical calculations.

Conclusion For Small Dataset: For small dataset the test assigned to me was "CS205_small_testdata__31". It had 10 features and 300 instances. The best feature subset obtained was "[8, 1, 10]" with the accuracy of "94.33%" for forward selection algorithm. The best feature obtained for backward elimination was "[1, 2, 8]", which has an accuracy of 93.33%. For this problem, backward elimination is not as accurate as forward selection. However, because it does find features '8'and '1' as its best feature subsets, it gives us confidence that feature '8' and '1' are really a good feature.

In Figure 1 we see the result of running forward selection on CS205_small_testdata__31.txt, which was the small file assigned to me.

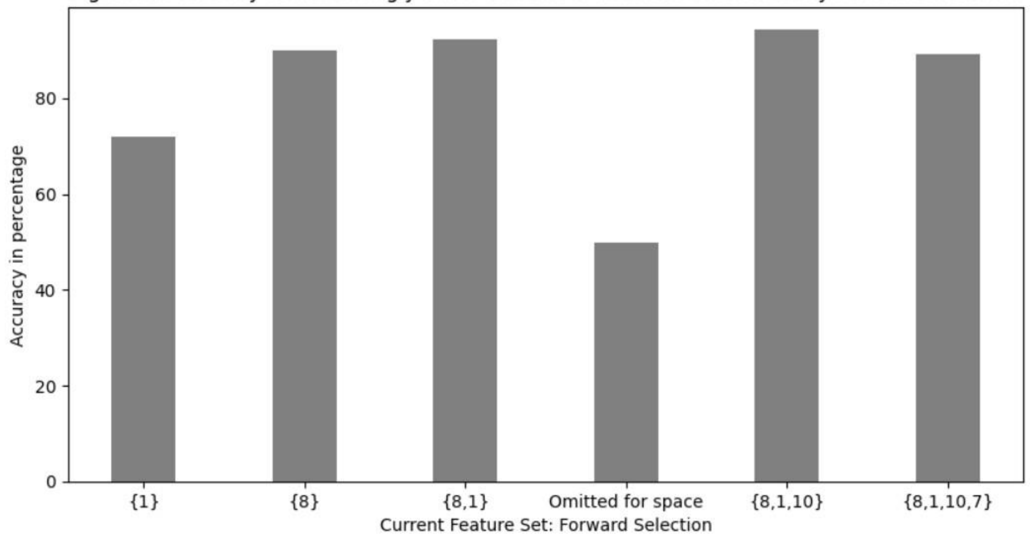
Figure 1: Accuracy of increasingly small subsets of features discovered by forward selection.

Conclusion for Large Dataset: For large dataset the test assigned to me was "CS205_large_testdata__41". It had 250 features and 900 instances. The best feature subset obtained was "[193, 33]" with the accuracy of "95.77%".

In Figure 2 we see the result of running forward selection on CS205_large_testdata__41.txt, which was the large file assigned to me.

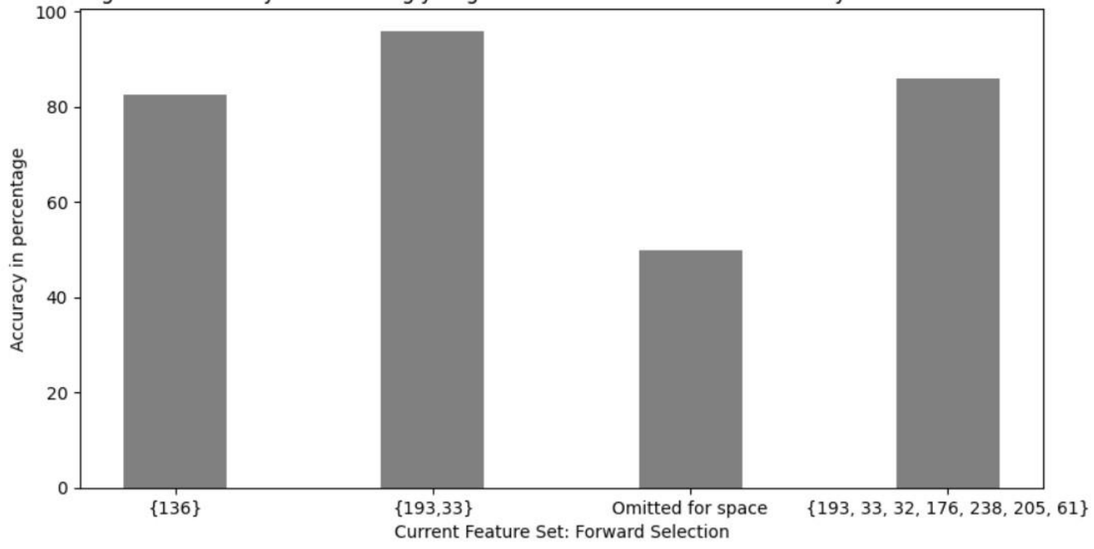
Figure 2: Accuracy of increasingly large subsets of features discovered by forward selection.

Computational effort for search: I implemented the search in Jupyter Notebook and ran all the tests on Google Colab (a cloud resource by google) instead of running them locally. The running time for the tests are depicted in the Table 1.

	Small dataset (10 features, 300 instances)	Large dataset (250 features, 900 instances)
Forward Selection	7 seconds	3.5 hours
Backward Elimination	9 seconds	4.3 hours

Table 1: Time duration to run the tests.

Trace: Below I show a single trace of my algorithm. I am *only* showing Forward Selection on the small dataset.

```
Welcome to Prince's Feature Selection Algorithm

Type the file name to test:CS205_small_testdata__31.txt

Type the number corresponding to algorithm to use it:

1)Forward Selection

2)Backward elimination

Enter choice here: 1

This dataset has 10 features(not including class attributes), with 300 instances.

Normalizing dataset

Normalization done!

Beginning Search.

    Using feature(s) {1} accuracy is 72.33%

    Using feature(s) {2} accuracy is 69.0%

    Using feature(s) {3} accuracy is 70.33%

    Using feature(s) {4} accuracy is 67.67%

    Using feature(s) {5} accuracy is 68.33%

    Using feature(s) {6} accuracy is 70.0%

    Using feature(s) {7} accuracy is 68.33%

    Using feature(s) {8} accuracy is 90.0%

    Using feature(s) {9} accuracy is 71.67%

    Using feature(s) {10} accuracy is 69.0%

Feature set [8], was best, accuracy is :90.0%

    Using feature(s) {1} accuracy is 92.33%

    Using feature(s) {2} accuracy is 89.33%

    Using feature(s) {3} accuracy is 86.67%
    Using feature(s) {4} accuracy is 86.0%
    Using feature(s) {5} accuracy is 86.0%
    Using feature(s) {6} accuracy is 87.67%
    Using feature(s) {7} accuracy is 84.33%
    Using feature(s) {9} accuracy is 86.67%
    Using feature(s) {10} accuracy is 88.0%
Feature set [8, 1], was best, accuracy is :92.33%
    Using feature(s) {2} accuracy is 93.33%
    Using feature(s) {3} accuracy is 89.0%
    Using feature(s) {4} accuracy is 89.0%
    Using feature(s) {5} accuracy is 90.67%
    Using feature(s) {6} accuracy is 89.33%
    Using feature(s) {7} accuracy is 88.33%
    Using feature(s) {9} accuracy is 92.0%
    Using feature(s) {10} accuracy is 94.33%
Feature set [8, 1, 10], was best, accuracy is :94.33%
    Using feature(s) {2} accuracy is 88.0%
    Using feature(s) {3} accuracy is 87.67%
    Using feature(s) {4} accuracy is 85.33%
    Using feature(s) {5} accuracy is 87.0%
    Using feature(s) {6} accuracy is 88.0%
    Using feature(s) {7} accuracy is 89.33%
    Using feature(s) {9} accuracy is 87.67%
Feature set [8, 1, 10, 7], was best, accuracy is :89.33%
    Using feature(s) {2} accuracy is 84.33%
    Using feature(s) {3} accuracy is 82.33%
    Using feature(s) {4} accuracy is 85.33%
    Using feature(s) {5} accuracy is 84.67%
    Using feature(s) {6} accuracy is 84.0%
    Using feature(s) {9} accuracy is 84.67%
Feature set [8, 1, 10, 7, 4], was best, accuracy is :85.33%
```

```
    Using feature(s) {2} accuracy is 82.33%

    Using feature(s) {3} accuracy is 80.33%

    Using feature(s) {5} accuracy is 84.0%

    Using feature(s) {6} accuracy is 80.67%

    Using feature(s) {9} accuracy is 80.0%

Feature set [8, 1, 10, 7, 4, 5], was best, accuracy is :84.0%

    Using feature(s) {2} accuracy is 82.67%

    Using feature(s) {3} accuracy is 79.33%

    Using feature(s) {6} accuracy is 80.0%

    Using feature(s) {9} accuracy is 79.67%

Feature set [8, 1, 10, 7, 4, 5, 2], was best, accuracy is :82.67%

    Using feature(s) {3} accuracy is 77.33%

    Using feature(s) {6} accuracy is 78.0%

    Using feature(s) {9} accuracy is 77.67%

Feature set [8, 1, 10, 7, 4, 5, 2, 6], was best, accuracy is :78.0%

    Using feature(s) {3} accuracy is 76.0%

    Using feature(s) {9} accuracy is 74.33%

Feature set [8, 1, 10, 7, 4, 5, 2, 6, 3], was best, accuracy is :76.0%

    Using feature(s) {9} accuracy is 72.0%

Feature set [8, 1, 10, 7, 4, 5, 2, 6, 3, 9], was best, accuracy is :72.0%

Finished Search!! Best feature subset is: [8, 1, 10], which has an accuracy of 94.33%
```

Code: Below is my code for this project, you can also find it in the GitHub.

https://github.com/princerokn/CS205

```python
import math
import re
import operator
import decimal
def nearestNeighbor(train,test,current):
    Distances = []
    for t in train:
        distance = 0
        for i in current:
            distance += math.pow(t[i] - test[i], 2)
        d = math.sqrt(distance)
        Distances.append([d,t[0]])
    Distances.sort(key=operator.itemgetter(0))
    return  Distances[0][1]
def leaveOneOutCrossValidaation(graph,currFeature,feature,algo):
    cur = []
    count = 0
    for i in currFeature:
        cur.append(i)
    if algo == 1:
        cur.append(feature)
    else:
        cur.remove(feature)
    for j in range(0, len(graph)):
        train = graph[:]
        test = train.pop(j)
        gr = nearestNeighbor(train,test,cur)
        if test[0] == gr:
            count += 1
    accuracy = count / float(len(graph))
    return accuracy
def forwardSelection(graph):
    featureNo = len(graph[0])
    accuray = []
    finalFeatures = []
    finalAccuracy = 0
    currFeature = []
    print ("Beginning Search.")
    for i in range(1,featureNo):
        bestAcc = 0
        featureToAdd = -1
        for j in range(1,featureNo):
            if j not in currFeature:
                accuracy =
leaveOneOutCrossValidaation(graph,currFeature,j,1)
                print ("     Using feature(s) {"+str(j)+"} accuracy is
"+str(round((accuracy * 100),2))+"%")
                if accuracy > bestAcc:
                    bestAcc = accuracy
                    featureToAdd = j
        if featureToAdd != -1:
            currFeature.append(featureToAdd)
            print ("Feature set "+ str(currFeature)+", was best, accuracy
is :"+str(round((bestAcc * 100),2))+"%")
            accuray.append(bestAcc)
        if bestAcc > finalAccuracy:
            finalFeatures = currFeature[:]
            finalAccuracy = bestAcc
    print ("Finished Search!! Best feature subset is:
"+str(finalFeatures)+", which has an accuracy of
"+str(round((finalAccuracy*100),2))+"%")
def backwardElimination(graph):
    finalFeatures = []
    finalAccuracy = 0
    currFeature = []
    for i in range (1,len(graph[0])):
        currFeature.append(i)
    featureNo = len(graph[0])
    accuray = []
    print ("Beginning Search.")
    for i in range(1, featureNo):
        featureToRemove = -1
        bestAcc = 0
        for j in range(1, featureNo):
            if j in currFeature:
                accuracy = leaveOneOutCrossValidaation(graph,
currFeature,j,2)
                print ("     Using feature(s) {"+str(j)+"} accuracy is
"+str(round((accuracy * 100),2))+"%")
                if accuracy > bestAcc:
                    bestAcc = accuracy
                    featureToRemove = j
        if featureToRemove != -1:
            currFeature.remove(featureToRemove)
            print("Feature set " + str(currFeature) + ", was best, accuracy
is :" + str(round((bestAcc * 100), 2)) + "%")
            accuray.append(bestAcc)
        if bestAcc > finalAccuracy:
            finalFeatures = currFeature[:]
            finalAccuracy = bestAcc
    print ("Finished Search!! Best feature subset is:
"+str(finalFeatures)+", which has an accuracy of
"+str(round((finalAccuracy*100),2))+"%")
def dataNormalization(graph):
   dataSet = graph
   avg = [0.00]*(len(dataSet[0])-1)
   sd = [0.00]*(len(dataSet[0])-1)
   for i in dataSet:
      for j in range (1,(len(i))):
         avg[j-1] +=  i[j]
   for i in range(len(avg)):
      avg[i] = (avg[i]/len(dataSet))
   for i in dataSet:
      for j in range (1,(len(i))):
         sd[j-1] +=  pow((i[j] - avg[j-1]),2)
   for i in range(len(sd)):
      sd[i] = math.sqrt(sd[i]/len(dataSet))
   for i in range(len(dataSet)):
      for j in range (1,(len(dataSet[0]))):
         dataSet[i][j] = (dataSet[i][j] - avg[j-1])/sd[j-1]
   return dataSet
if __name__ == '__main__':
    print("Welcome to Prince's Feature Selection Algorithm")
    testfile = input("Type the file name to test:")
    with open(testfile) as file:
        dataset = file.readlines()
    rows = []
    graph = []
    dataset = [i.strip() for i in dataset]
    for line in dataset:
        values = re.split(" +",line)
        for val in values:
            value = float(decimal.Decimal(val))
            rows.append(value)
        graph.append(rows)
        rows = []
    print ("Type the number corresponding to algorithm to use it:
\n1)Forward Selection \n2)Backward elimination")
    n = int(input("Enter choice here: "))
    print ("This dataset has "+str(len(graph[0])-1)+" features(not
including class attributes), with "+str(len(graph))+" instances.")
```

```python
    print ("Normalizing dataset")
    graph = dataNormalization(graph)
    print("Normalization done!")
    start_time = time.time()
    if n==1:
        forwardSelection(graph)
    elif n==2:
        backwardElimination(graph)
    else:
        print("You have entered wrong choice")
```