Project - 2

CS205 - Artificial Intelligence **Prof. Eamonn Keogh**

Name: Harsha Paruchuri
SID#: 862122817
Email: hparu001@ucr.edu
Date: Monday, March 18, 2019

For the completion of this project, I referred to the following sources:

- http://www.cs.ucr.edu/~eamonn/205/ : Slides on machine learning, evaluation
- http://www.d.umn.edu/~deoka001/Normalization.html
- https://docs.python.org/3/tutorial/floatingpoint.html
- https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-fromscratch/

The results, observations and conclusions obtained in this project are my work only.
All the key code components were written by me and some trivial code snippets were written by referring to the sources above.

Introduction

This is a feature selection application developed as a project in the CS205 – Artificial Intelligence course at the University of California Riverside by Prof. Eamonn Keogh. The application provides a choice of three algorithms namely: Forward Search, Backward Elimination and a Speed Up. The goal of the application is to select those features that can determine the class to which the record belongs, with the highest possible accuracy. The algorithm at the core of the application is the K - Nearest Neighbors which would be trained and tested in a leave-one-out manner.

Dataset

The datasets available for this application are:
CS205_SMALLtestdata_17.txt
CS205_BIGtestdata_10.txt.
Both datasets have two class labels 1 and 2. The SMALL dataset has ten features and a hundred instances. The BIG dataset has fifty features and hundred instances. The data is in IEEE 754 floating point format. The first column is the class label and the subsequent columns are the features. The elements are space - separated.

Preprocessing

On input, each value in the dataset was converted to a python floating point value. The class labels were extracted into a one-dimensional array, leaving the feature values in a two-dimensional array. This feature matrix was sent to a normalization function, normalize(activeDataSet).This function calculates the mean value of a feature and the standard
deviation for the same. The normalized value is calculated by:
Normalized_Val = (Value - Mean(Feature)) / Standard_deviation(Feature) i.e.
dataSet[i][j] = (dataSet[i][j] - average[j-1])/ stds[j-1]
The result of this function would be values inside each of the features normalized to values between 0 and 1.

Algorithms

Leave - one - out Cross Validation

In order to find how accurately a set of features would classify a record into one of the two classes, the KNN algorithm is used. Here, the euclidean distance between the values of the respective features between the training and the test data is computed. These distances are then sorted in ascending order and the top “K” points in the training set are picked and returned. For the purpose of this application, “K” is taken as one. The label returned by the KNN method is then compared with the label of the test data. The accuracy is calculated by dividing the number of correct predictions with the number of records in the dataset. The percentage accuracy is returned to the function calling this algorithm which is then utilized as described below.

Forward Search

The forward search algorithm in involves, starting with an empty feature set and checking the accuracy by including each feature. This algorithm follows a level by level approach in which the first level would attempt to include each one of the features individually and pick the feature which results in the highest accuracy of class label prediction. In the next level, one more feature is picked from the remaining features in the dataset, and the prediction accuracy for the combined feature set is calculated. In this way, the best feature set in each level is selected. Finally, from among the feature sets picked at each level, the one with the highest accuracy is selected as the best feature set for the given dataset.

Backward Elimination

The Backward elimination is a reverse approach to the forward search. Here the application starts with all the features as being correct and the accuracy is calculated. In the first level, each individual feature is selected and removed, the prediction accuracy for the subsequent features is calculated and the best one is picked. In the next levels, further features are removed. Finally, the feature set with the best accuracy is selected as the features that accurately indicate the class of the records in the dataset.

Speed Up Algorithm

The special search algorithm is a faster variant of the forward search. The faster performance is achieved by pruning those features that perform worse than the feature. In this method, the best accuracy computed till that point is sent to the nearest neighbors function. This best accuracy indicates the number of incorrect class labels predicted by the best feature set found till that point. Therefore, as soon as the current feature set exceeds that number of incorrect predictions, it is discarded and accuracy is set to zero. This results in an approximately 30% faster performance.

Results

Dataset: CS205_SMALLtestdata_17.txt

Forward Search:

Best feature subset was [4, 6] , accuracy is 94.0
Time Taken: 3.8141677379608154 seconds

Backward Elimination:

Best feature subset was [4,6] , accuracy is 86.5
Time Taken: 5.0016772747039795 seconds

SpeedUp Search:

Best feature subset was [4,6] , accuracy is 96.5
Time Taken: 3.2161881923675537 seconds

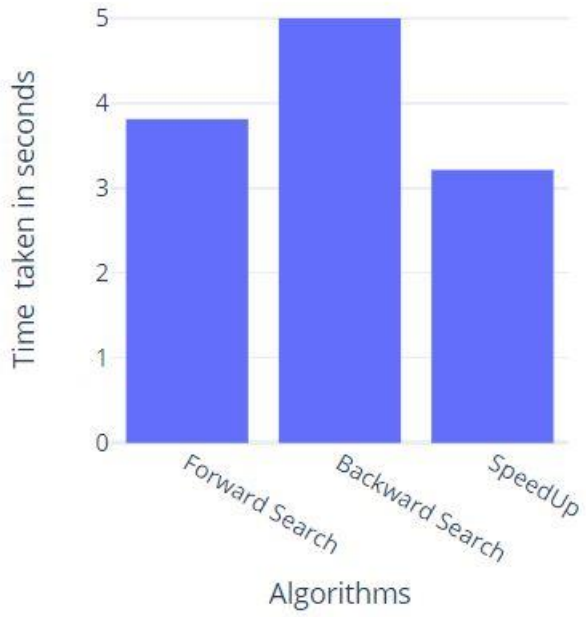

Fig. 1.1 Time taken by each algorithm for the SMALLtestdata_17.

Dataset: CS205_BIGtestdata_10.txt

Forward Search:

Best feature subset was [2, 4] , accuracy is 93.0
Time Taken: 2280.1005494815940438 seconds

Backward Elimination:

Best feature subset was [2,4] , accuracy is 84.0
Time taken: 2754.0329688956426253 seconds

SpeedUp Search:

Best feature subset was [2,4], accuracy is 94.0
Time taken: 503.03700971603394 seconds

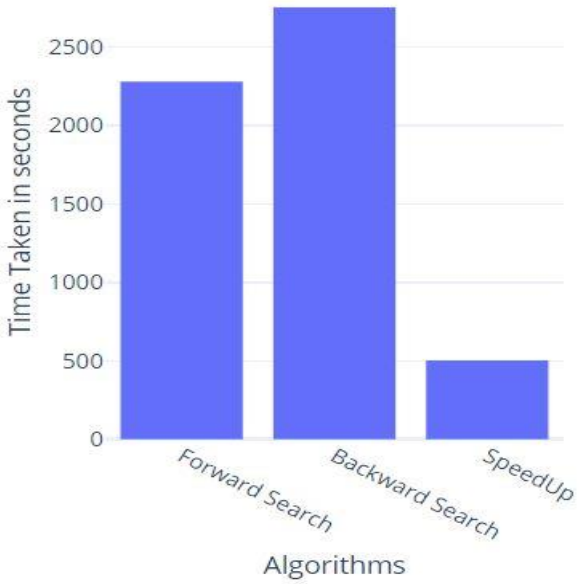

Fig. 1.2 Time taken by each algorithm for the LARGEtestdata_10.

Accuracy Comparison

Algorithm	Small Dataset Accuracy	Large Dataset Accuracy
Forward Search	94%	93%
Backward Search	86.5%	84%
Speed Up	96.5%	94%

Table 1: Comparison of Accuracies

In the figure below X-axis: algorithms, Y-axis: Small dataset accuracies(in percentage), Z-axis: Large dataset accuracies(in percentage).

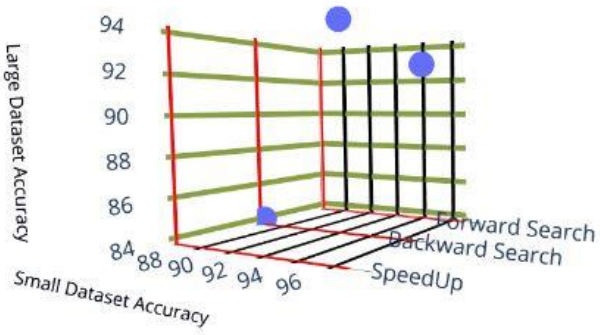

Fig. 1.3. Accuracy of the best feature in each algorithm

Conclusion

This application illustrates very clearly, the differences in performance and accuracy observed when the approach to the feature selection problem is different. As seen in fig. 1.3, and table 1 the accuracy of the backward elimination algorithm is slightly lower when compared to the forward search and the speed up search algorithms. Also, as seen in fig 1.1 and fig 1.2, Backward elimination takes more time than the other two algorithms Different feature results were obtained when the "K" value was increased to 3 for the backward elimination. From fig. 1.3 it can be seen that with the bigger dataset the accuracy drops slightly. Another observation (from fig.1.1 and fig. 1.2) is that the speed up algorithm which prunes the evaluation of the unnecessary feature sets, they perform faster, and with almost same results and accuracy as the forward search. Since it clearly outperforms both the algorithms it can be concluded that forward searching with pruning is the better technique for feature selection using KNN.

Output Trace

The output trace for the forward search and speedup search with the SMALLtestdata_17 dataset is shown below:

Forward Search Trace

Enter the algorithm:

1)Forward Selection

2)Backward Selection

3)Original Algorithm

Enter your choice: 1

This dataset has 10 features(not including class attributes),with 200 instances.

Please wait While I normalize data.....Done!

Beginning Search

Using feature(s) [1] accuracy is 0.67

Using feature(s) [2] accuracy is 0.635

Using feature(s) [3] accuracy is 0.685

Using feature(s) [4] accuracy is 0.865

Using feature(s) [5] accuracy is 0.69

Using feature(s) [6] accuracy is 0.685

Using feature(s) [7] accuracy is 0.71

Using feature(s) [8] accuracy is 0.715

Using feature(s) [9] accuracy is 0.7

Using feature(s) [10] accuracy is 0.705

On level 1 I added feature 4 to current set

Feature set [4] was best with accuracy:0.865

Using feature(s) [1] accuracy is 0.805

Using feature(s) [2] accuracy is 0.75

Using feature(s) [3] accuracy is 0.86

Using feature(s) [5] accuracy is 0.85

Using feature(s) [6] accuracy is 0.94

Using feature(s) [7] accuracy is 0.84

Using feature(s) [8] accuracy is 0.825

Using feature(s) [9] accuracy is 0.855

Using feature(s) [10] accuracy is 0.785

On level 2 I added feature 6 to current set

Feature set [4, 6] was best with accuracy:0.94

Using feature(s) [1] accuracy is 0.865

Using feature(s) [2] accuracy is 0.865

Using feature(s) [3] accuracy is 0.9

Using feature(s) [5] accuracy is 0.895

Using feature(s) [7] accuracy is 0.86

Using feature(s) [8] accuracy is 0.905

Using feature(s) [9] accuracy is 0.91

Using feature(s) [10] accuracy is 0.9

On level 3 I added feature 9 to current set

Feature set [4, 6, 9] was best with accuracy:0.91

********** REMOVED FEW LINES**********
**********TO SAVE PAPER**********

Feature set [4, 6, 9, 1, 8, 5, 10, 2] was best with accuracy:0.77

Using feature(s) [3] accuracy is 0.735

Using feature(s) [7] accuracy is 0.725

On level 9 I added feature 3 to current set

Feature set [4, 6, 9, 1, 8, 5, 10, 2, 3] was best with accuracy:0.735

Using feature(s) [7] accuracy is 0.71

On level 10 I added feature 7 to current set

Feature set [4, 6, 9, 1, 8, 5, 10, 2, 3, 7] was best with accuracy:0.71

Finished Search!!!

Best feature subset is:

[4, 6]

Best accuracy is:

0.94

--- 3.8141677379608154 seconds ---

Speed-Up search trace

Enter your choice: 3

This dataset has 10 features(not including class attributes),with 200 instances.

Please wait While I normalize data.....Done!

On level 1 I added feature 4 to current set

On level 2 I added feature 6 to current set

Final set of features are:

[4, 6]

Final accuracy is:

0.965

--- 3.2161881923675537 seconds ---

Code

```
"""
Created on Wed Mar  6 22:51:52 2019

@author: galahad
"""

import decimal
import re
import operator
import math
import time

def euclidean(trn,tst,curr):
    dist = 0
    for i in curr:
        dist += math.pow(trn[i]-tst[i],2)
    return math.sqrt(dist)

def KNN(train,test,curr):
    D = []
    for r in train:
        dist = euclidean(r,test,curr)
        D.append([dist,r[0]])
    D.sort(key=operator.itemgetter(0))
    return D[0][1]

def leave_one_out_cross_validation(table,current_feat,feature,choice):

    freq = 0
    current = current_feat[:]
    if choice == 1:
        current.append(feature)
    else:
        current.remove(feature)

    for k in range(0, len(table)):
        train = table[:]
        test = train.pop(k)
        group = KNN(train,test,current)
        if test[0] == group:
            freq += 1
    acc = freq / float(len(table))
    return acc
```

```
def forward_selection(table):

    final_features = []
    final_accuracy = 0
    current_features = []
    feature_size = len(table[0])
    accuracies = []
    print("Beginning Search")
    for i in range(1,feature_size):
        #print " On the "+str(i)+"th level of the tree"
        feature_to_add_this_level = -1
        best_accuracy = 0

        for j in range(1,feature_size):
            if j not in current_features:
                #print " Considering adding the "+str(j)+"th feature"
                accuracy = leave_one_out_cross_validation(table,current_features,j,1)
                print ("     Using feature(s) ["+str(j)+"] accuracy is "+str(accuracy))
                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    feature_to_add_this_level = j

        if feature_to_add_this_level != -1:
            current_features.append(feature_to_add_this_level)
            print ("On level "+str(i)+" I added feature "+ str(feature_to_add_this_level) + " to current
set")
            print ("Feature set "+ str(current_features)+" was best with
accuracy:"+str(best_accuracy))
            accuracies.append(best_accuracy)
        if best_accuracy > final_accuracy:
            final_features = current_features[:]
            final_accuracy = best_accuracy
    print ("Finished Search!!!")
    print ("Best feature subset is: ")
    print (final_features)
    print ("Best accuracy is: ")
    print (final_accuracy)

def leave_one_out_cross_validation_speedup(table,current_feat,feature,best_so_far):

    local_miss = 0
    freq = 0
    current = current_feat[:]
    current.append(feature)
    for k in range(0, len(table)):
        train = table[:]
```

```
        test = train.pop(k)
        group = KNN(train,test,current)
        if test[0] == group:
            freq += 1
        else:
            local_miss += 1
            if local_miss > best_so_far:
                return [0,best_so_far]
    best_so_far = local_miss
    acc = freq / float(100)
    return [acc,best_so_far]

def backward_elimination(table):
    final_features = []
    final_accuracy = 0
    current_features = list(range(1,len(table[0])))
    feature_size = len(table[0])
    accuracies = []
    print ("Beginning Search")
    for i in range(1, feature_size):
        # print "On the "+str(i)+"th level of the tree"
        feature_to_remove_this_level = -1

        #best_accuracy = leave_one_out_cross_validation(table,current_features,-1,2)
        best_accuracy = 0
        for j in range(1, feature_size):
            if j in current_features:
                # print "Considering adding the "+str(j)+"th feature"
                accuracy = leave_one_out_cross_validation(table, current_features,j,2)
                print ("     Using feature(s) [" + str(j) + "] accuracy is " + str(accuracy))
                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    feature_to_remove_this_level = j

        if feature_to_remove_this_level != -1:
            current_features.remove(feature_to_remove_this_level)
            print ("On level " + str(i) + " I removed feature " + str(feature_to_remove_this_level) + "
from current set with accuracy:" + str(best_accuracy))
            accuracies.append(best_accuracy)
        if best_accuracy > final_accuracy:
            final_features = current_features[:]
            final_accuracy = best_accuracy

    print ("Finished Search!!!")
    print ("Best feature subset is: ")
    print (final_features)
```

```
    print ("Best accuracy is: ")
    print (final_accuracy)

def speedup(table):

    best_so_far=100
    final_features = []
    final_accuracy = 0
    current_features = []
    feature_size = len(table[0])

    for i in range(1,feature_size):
        #print "On the "+str(i)+"th level of the tree"
        feature_to_add_this_level = -1
        best_accuracy = 0

        for j in range(1,feature_size):
            if j not in current_features:
                #print "Considering adding the "+str(j)+"th feature"
                result = leave_one_out_cross_validation_speedup(table,current_features,j,best_so_far)
                accuracy = result[0]
                best_so_far = result[1]
                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    feature_to_add_this_level = j

        if feature_to_add_this_level != -1:
            current_features.append(feature_to_add_this_level)
            print ("On level "+str(i)+" I added feature "+ str(feature_to_add_this_level) + " to current
set")

        if best_accuracy > final_accuracy:
            final_features = current_features[:]
            final_accuracy = best_accuracy

    print ("Final set of features are: ")
    print (final_features)
    print ("Final accuracy is: ")
    print (final_accuracy)
def normalize(activeDataSet):
            dataSet = activeDataSet
            average = [0.00]*(len(dataSet[0])-1)
            stds = [0.00]*(len(dataSet[0])-1)
            #           get averages
            for i in dataSet:
                        for j in range (1,(len(i))):
```

```
                        average[j-1] +=  i[j]
        for i in range(len(average)):
                average[i] = (average[i]/len(dataSet))
        #       get std's sqrt((sum(x-mean)^2)/n)
        for i in dataSet:
                for j in range (1,(len(i))):
                        stds[j-1] +=  pow((i[j] - average[j-1]),2)
        for i in range(len(stds)):
                stds[i] = math.sqrt(stds[i]/len(dataSet))
        #       calculate new values (x-mean)/std
        for i in range(len(dataSet)):
                for j in range (1,(len(dataSet[0]))):
                        dataSet[i][j] = (dataSet[i][j] - average[j-1])/ stds[j-1]
        return dataset
if __name__ == '__main__':
    file = input( )
    with open(file) as f:
        data = f.readlines()
    row = []
    table = []
    data = [x.strip() for x in data]
    for line in data:
        values = re.split(" +",line)
        for v in values:
            val = float(decimal.Decimal(v))
            row.append(val)
        table.append(row)
        row = []
    table = normalize(table)
    print ("""Enter the algorithm: \n
        1)Forward Selection \n
        2)Backward Selection\n
        3)Original Algorithm
        """)
    n = int(input("Enter your choice: "))
    print ("This dataset has "+str(len(table[0])-1)+" features(not including class attributes),with
"+str(len(table))+" instances.")
    print ("Please wait While I normalize data.....Done!")
    start_time = time.time()
    if n==1:
        forward_selection(table)
    elif n==2:
        backward_elimination(table)
    elif n==3:
        speedup(table)
    print("--- %s seconds ---" % (time.time() - start_time))
```