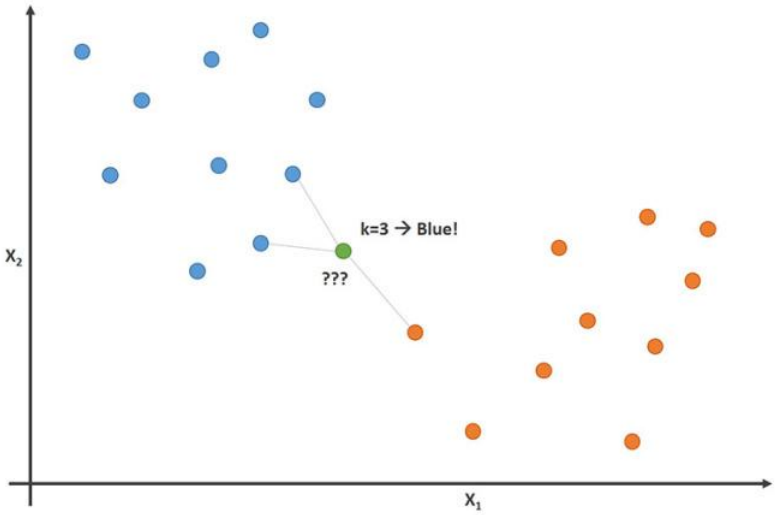
CS 205 - Artificial Intelligence

Dr. Eamonn Keogh

Project -2 : Feature Selection with Nearest Neighbor



HANISHA SREE SANGAM

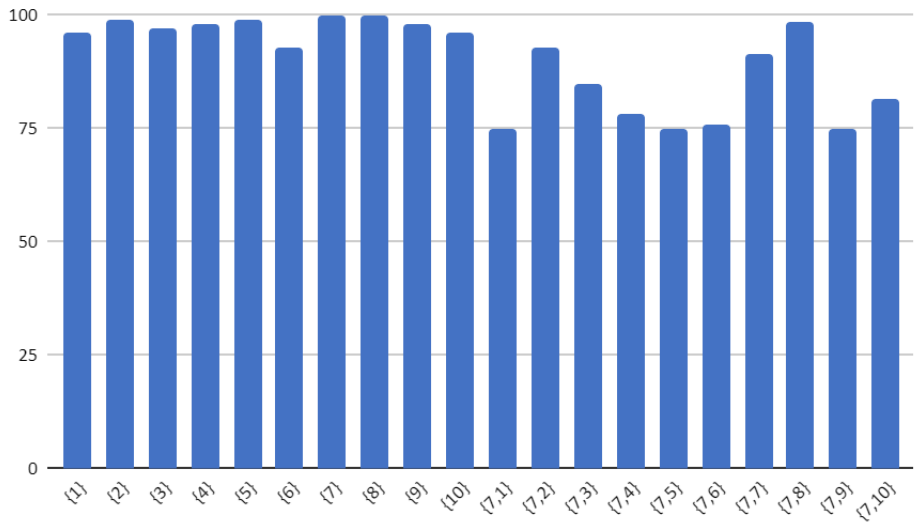
06/10/2021.

SPRING 2021
(SID - 862191473)

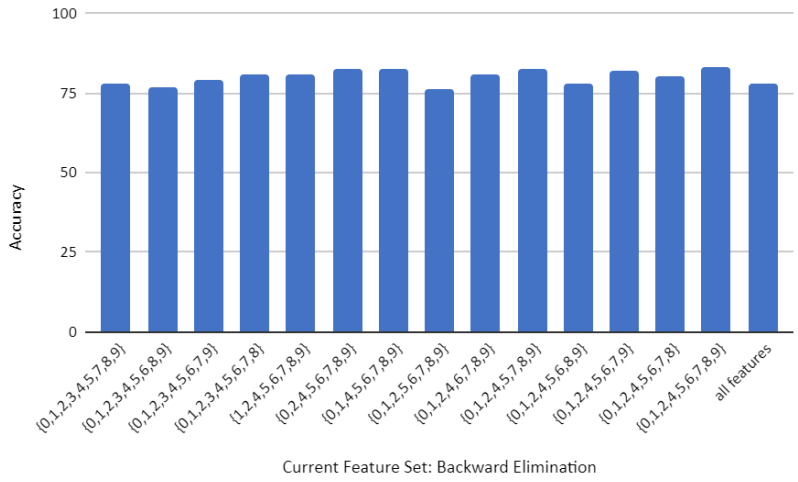
INTRODUCTION

Implementing Feature Selection with Nearest Neighbor using forward selection and backward selection is the Project in CS 205- Artificial Intelligence course, under the Professor - Dr. Eamonn Keogh at University of California, Riverside for the Spring 2021 quarter. This Report is a detailed explanatory of my work.

Problem Statement : Implementing Feature Selection with Nearest Neighbor using forward selection and backward selection



In the above figure we can see accuracies of various feature sets formed due to the implementation of forward selection on CS205_small_testdata__1.txt.



In the above figure we can see accuracies of various feature sets formed due to the implementation of backward selection on CS205_small_testdata__1.txt.

DataSets Used :
Small datasets -CS205_small_testdata__1.txt.
Large datasets - CS205_large_testdata__44.txt.

Implementation of the code step wise :

- We start by loading the data. The files were in txt format. Those files had to be modified to csv which made it easier to load as a dataframe in python.
- I added headers to easily read the data in the files.
- I split the data into train and test sets. python gives us a function to do that, we mention the size of the test set and it splits accordingly.

```
X, testX, Y, testY = train_test_split(x, y, test_size=1 / 3, random_state=0)
model = NearestNeighborsClassifier(k=15) #creating an object of class NearestNeighborsClassifier
```

- For the first classification, I selected all rows and all columns of the dataset,then we applied our function to classify and compute accuracy.

```
#running classification on all features:
prediction = model.classify(testX, k=15)

#computing accuracy for all features:
correctCount = 0
count = 0
for count in range(np.size(prediction)):
    if testY[count][0] == prediction[count]:
        correctCount = correctCount+1
    count = count + 1

print("Accuracy on test set for all features by our model      :  ", (correctCount / count) * 100)
```

- For Forward feature selection we make an array called accuracies that will save the accuracies of all the individual features. And then loop over each feature.

```
i = 1
data2 = data.iloc[:, :]
allFeatures = data2.columns
accuracies = []
```

- This line of code selects all rows of the columns 0(which is target) and ith feature. We initiated i at 1, which means it started with Feature 1 and will continue to iterate till the last feature.

```
for counter in range(np.size(allFeatures) - 1):
    newData = data2.iloc[:, [0, i]]
    #print(newData.columns)
    i = i + 1
    a = newData.iloc[:, :].values
    b = newData.iloc[:, :].values
    A, testA, B, testB = train_test_split(a, b, test_size=1 / 3, random_state=0)
    # print("A", A.shape)
```

- At each iteration, once we select the columns, we repeat the classification process

Data selection (Columns in this case)
train/test data split
store training data
classify (run model)
compute accuracy.
we save the accuracy of the ith feature in accuracies

```
newData = data2.iloc[:, [0, i]]
#print(newData.columns)
i = i + 1
a = newData.iloc[:, :].values
b = newData.iloc[:, :].values
A, testA, B, testB = train_test_split(a, b, test_size=1 / 3, random_state=0)
# print("A", A.shape)
model.storeTrainingData(A, B)
predicting = model.classify(testA, k=15)
# Accuracy computation
Counter = 0
counting = 0
for counting in range(np.size(predicting)):
    if testB[counting][0] == predicting[counting]:
        Counter = Counter + 1
    counting = counting + 1
accuracy = (Counter / counting) * 100
accuracies.append(accuracy)
```

- We use max function to get the maximum accuracy value.

```
best_feature_value=max(accuracies) #value of the best accuracy found,
```

- we find out the index of that value to figure out the corresponding feature. we add one to the index value because accuracy for Feature 1 was stored at index 0 of accuracies and so on.

```
best_feature_index=accuracies.index(best_feature_value)+1   #the index/feature number with that accuracy
```

- Once we have the best feature with highest accuracy value, we use a loop to combine the

best feature with the rest of the features.

- Code for combinations

```
#using the individual best feature to combine it further
t=1
combination_accuracies=[]
for counter2 in range(np.size(allFeatures) - 1):
    newData = data2.iloc[:, [0, best_feature, t]]
    #print(newData.columns)
    a = newData.iloc[:, :].values
    b = newData.iloc[:, :].values
    A, testA, B, testB = train_test_split(a, b, test_size=0.2, random_state=0)
    #print("A", A.shape)
    model.storeTrainingData(A, B)
    predicting = model.classify(testA, k=15)
    t = t + 1
```

- we make combinations, and then just as we saved accuracy values of independent features in array 'accuracies', for combinations we save them in a list called combination_accuracies.

```
# Accuracy computation of combinations of features
Counter = 0
counting = 0
for counting in range(np.size(predicting)):
    if testB[counting][0] == predicting[counting]:
        Counter = Counter + 1
    counting = counting + 1
acc=(Counter / counting) * 100
print("Accuracy for subset: ",newData.columns,"is: ", acc)
combination_accuracies.append(acc)
```

- Finding the best combination by using max function and finding its index.

```
#selecting the best combination:
best_combo_value=max(combination_accuracies)
best_combo_index=combination_accuracies.index(best_combo_value)+1
#print(best_combo_index)
```

- The next bit is a little different, instead of getting to further combinations right away, we check if the combinations improved the accuracy at all, if it did, we will continue by taking the best combination and looping it further to add a third feature to the subset. but in case the accuracy value dropped, we prompt the user with the subset with maximum accuracy so far.

```
if(best_combo_value>best_feature_value): #check if the accuracy of the model is improved by the new subset
                                         # if yes, combine it further, if no, present it as the best combination
    t = 1
```

- If the condition results 'true', We take the best combination and make further subsets using the loop for the third Feature in the set. repeat the classification process by selecting columns, train/test split, model execution, accuracy computation.

```
if(new_best_combo_value>best_combo_value): #check if the accuracy of the model is improved by the new subset
                                           # if yes, combine it further, if no, present it as the best combination
    t = 1
    new_combination_accuracies = []
    for counter4 in range(np.size(allFeatures) - 1):

        newData = data2.iloc[:, [0, best_feature, best_combo_index,new_best_combo_index, t]]
        #print(newData.columns)
        a = newData.iloc[:, :].values
        b = newData.iloc[:, :].values
        A, testA, B, testB = train_test_split(a, b, test_size=0.2, random_state=0)
        #print("A", A.shape)
        model.storeTrainingData(A, B)
        predicting = model.classify(testA, k=15)
        t = t + 1
```

- Otherwise, we print on screen the subset with best accuracy so far and move to backward elimination part.

```
else:
    print("The best feature subset is: Feature ", best_feature,", Feature:",
          best_combo_index, " With the accuracy: ", best_combo_value)
```

- Save the accuracies in another list, find max value and repeat the logic with an if else statement.

```
else:
    print("The best feature subset is: Feature ", best_feature,", Feature:",
          best_combo_index, " With the accuracy: ", best_combo_value)
```

- This logic is repeated until we have the best accuracy possible. at any point, if the accuracy is decreased, we go back to search for the maximum value we had and declare that as our best feature subset.

- For Backward Elimination, we start with all features and reduce one by one to check which subset gives us the best performance. we initiate i at all features which are all the columns in the feature set.

```
#Backward Elimination
print("\nBackward Feature Elimination\n")

Backward_accuracies=[]
data3 = data.iloc[:, :]
allFeatures = data3.columns
i = np.size(allFeatures)
# print(allFeatures)
cols=[]
```

- cols is an array that will store the subset of features at every iteration.

- We enter the loop, select all rows and all columns till ith column (which is all of them at first iteration), go through the classification process, compute accuracy, save it in backward_accuracies and move on to the next iteration in which i gets updated with one less feature. the loop continues until we have no features left.
- We then choose the best accuracy from backward_accuracies, find the index for corresponding subset in cols and give the user our results by printing it on screen.

```
backward_best=max(Backward_accuracies)
backward_index=Backward_accuracies.index(backward_best)



print("Best feature subset: ",cols[backward_index], "\nWith Accuracy: ", backward_best)
```

Output :

This is on Small dataset

Accuracy on test set for all features by our model : 83.0

Forward Feature Selection

Accuracy on Feature set Index(['Target', 'Feature 1'], dtype='object') by our model : 96.0
Accuracy on Feature set Index(['Target', 'Feature 2'], dtype='object') by our model : 99.0
Accuracy on Feature set Index(['Target', 'Feature 3'], dtype='object') by our model : 97.0
Accuracy on Feature set Index(['Target', 'Feature 4'], dtype='object') by our model : 98.0
Accuracy on Feature set Index(['Target', 'Feature 5'], dtype='object') by our model : 99.0
Accuracy on Feature set Index(['Target', 'Feature 6'], dtype='object') by our model : 93.0

Accuracy on Feature set Index(['Target', 'Feature 7'], dtype='object') by our model : 100.0
Accuracy on Feature set Index(['Target', 'Feature 8'], dtype='object') by our model : 100.0
Accuracy on Feature set Index(['Target', 'Feature 9'], dtype='object') by our model : 98.0
Accuracy on Feature set Index(['Target', 'Feature 10'], dtype='object') by our model : 96.0
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 1'], dtype='object') is: 75.0
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 2'], dtype='object') is: 93.33333333333333
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 3'], dtype='object') is: 85.0
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 4'], dtype='object') is: 78.33333333333333
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 5'], dtype='object') is: 75.0
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 6'], dtype='object') is: 76.66666666666667
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 7'], dtype='object') is: 91.66666666666666
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 8'], dtype='object') is: 98.33333333333333
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 9'], dtype='object') is: 75.0
Accuracy for subset: Index(['Target', 'Feature 7', 'Feature 10'], dtype='object') is: 81.66666666666667
The best feature subset is: Feature 7 With the accuracy: 100.0

Backward Feature Elimination

C (200, 11)
Back accuracy: 100.0
C (200, 10)
Back accuracy: 100.0
C (200, 9)
Back accuracy: 100.0
C (200, 8)
Back accuracy: 100.0
C (200, 7)
Back accuracy: 99.0
C (200, 6)
Back accuracy: 100.0
C (200, 5)
Back accuracy: 100.0

```
C (200, 4)
Back accuracy:  100.0
C (200, 3)
Back accuracy:  100.0
C (200, 2)
Back accuracy:  100.0
Best feature subset:  Index(['Target', 'Feature 1', 'Feature 2', 'Feature 3', 'Feature 4',
       'Feature 5', 'Feature 6', 'Feature 7', 'Feature 8', 'Feature 9',
       'Feature 10'],
      dtype='object')
With Accuracy:  100.0

Process finished with exit code 0
```

This is on Large dataset (Due to space limitation I could not include the complete result)

```
Accuracy on test set for all features by our model       :   79.66666666666666

Forward Feature Selection

Accuracy on Feature set Index(['Target', 'Feature 1'], dtype='object')  by our model
:   99.0
Accuracy on Feature set Index(['Target', 'Feature 2'], dtype='object')  by our model
:   99.66666666666667
Accuracy on Feature set Index(['Target', 'Feature 3'], dtype='object')  by our model
:   98.66666666666667
Accuracy on Feature set Index(['Target', 'Feature 4'], dtype='object')  by our model
:   100.0
……
……
……
……
……
Accuracy for subset:  Index(['Target', 'Feature 4', 'Feature 250'], dtype='object') is:
93.33333333333333
The best feature subset is: Feature  4  With the accuracy:  100.0

Backward Feature Elimination

C (600, 251)
Back accuracy:  80.0
C (600, 250)
Back accuracy:  80.33333333333333
……
```

```
.....
.....
.....
C (600, 2)
Back accuracy:  100.0
Best feature subset:  Index(['Target', 'Feature 1', 'Feature 2', 'Feature 3', 'Feature 4',
       'Feature 5', 'Feature 6'],
      dtype='object')
With Accuracy:  100.0

Process finished with exit code 0
```

Code :

Feature Selection using KNN:

```
#import math
import statistics
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split



class NearestNeighborsClassifier():

    def __init__(self, k):
        self.k = k

    def distance(self, a, b):
        return np.sqrt(np.sum(np.square(a - b)))  #formula for Euclidean Distance

    def storeTrainingData(self, X, Y): #Function that stores training data for
classification.
        self.X = X
        self.Y = Y
        trainSetInstance, trainSetFeatures= X.shape

    def classify(self, dataSet, k):
        self.dataSet = dataSet
        self.testInstance, self.testFeatures = dataSet.shape
        predict = np.zeros(self.testInstance)
        for i in range(self.testInstance):
            x = self.dataSet[i]
            neighbors = np.zeros(self.k)    #initializing the array that will store k
```

```
neighbors
            neighbors = self.findNeighbors(x, k)    #function call to get k nearest
neighbors
            predict[i] = statistics.mode(neighbors[:, 0])   #assinging the class with most
occurances in neighbors
        return predict

    def findNeighbors(self, x, k):
        distances = np.zeros(self.testInstance)
        for i in range(self.testInstance):
            d = self.distance(x, self.X[i])
            distances[i] = d
            temp = distances.argsort()
        YSorted = self.Y[temp]
        return YSorted[:self.k]


if __name__ == "__main__":

    # loading the data from csv file to dataframe
    data = pd.read_csv(r"small_testdata__1.csv")
    data = pd.read_csv(r"large_testdata.csv")

    ##normalizing the data
    normalized_data = (data - data.mean()) / data.std()
    ##another (optional) method for normalization of data
    #normalized_data = (data - data.min()) / (data.max() - data.min())

    #splitting the data into train and test sets. training dataset size: 2/3. test dataset
size:1/3.
    x = data.iloc[:, :-1].values
    y = data.iloc[:, :].values
    X, testX, Y, testY = train_test_split(x, y, test_size=1 / 3, random_state=0)
    model = NearestNeighborsClassifier(k=15) #creating an object of class
NearestNeighborsClassifier
    model.storeTrainingData(X, Y)

    #running classification on all features:
    prediction = model.classify(testX, k=15)

    #computing accuracy for all features:
    correctCount = 0
    count = 0
    for count in range(np.size(prediction)):
        if testY[count][0] == prediction[count]:
            correctCount = correctCount+1
```

```
        count = count + 1

    print("Accuracy on test set for all features by our model       :  ", (correctCount /
count) * 100)

  Forward Feature Selection :
 # Forward Feature Selection
    print("\nForward Feature Selection\n")


    #Start with computing model accuracies for each feature separately and save
them in a list called accuracies
    i = 1
    data2 = data.iloc[:, :]
    allFeatures = data2.columns
    accuracies = []
    for counter in range(np.size(allFeatures) - 1):
        newData = data2.iloc[:, [0, i]]
        #print(newData.columns)
        i = i + 1
        a = newData.iloc[:, :].values
        b = newData.iloc[:, :].values
        A, testA, B, testB = train_test_split(a, b, test_size=1 / 3, random_state=0)
        # print("A", A.shape)
        model.storeTrainingData(A, B)
        predicting = model.classify(testA, k=15)
        # Accuracy computation
        Counter = 0
        counting = 0
        for counting in range(np.size(predicting)):
            if testB[counting][0] == predicting[counting]:
                Counter = Counter + 1
            counting = counting + 1
        accuracy = (Counter / counting) * 100
        accuracies.append(accuracy)

        print("Accuracy on Feature set", newData.columns," by our model      :  ",
(Counter / counting) * 100)

    #selecting the feature with best individual accuracy for further combinations.

    best_feature_value=max(accuracies) #value of the best accuracy found,
    best_feature_index=accuracies.index(best_feature_value)+1   #the index/feature
number with that accuracy
    #print(best_feature_index)
    best_feature=best_feature_index
```

```
    #using the individual best feature to combine it further
    t=1
    combination_accuracies=[]
    for counter2 in range(np.size(allFeatures) - 1):
        newData = data2.iloc[:, [0, best_feature, t]]
        #print(newData.columns)
        a = newData.iloc[:, :].values
        b = newData.iloc[:, :].values
        A, testA, B, testB = train_test_split(a, b, test_size=0.2, random_state=0)
        #print("A", A.shape)
        model.storeTrainingData(A, B)
        predicting = model.classify(testA, k=15)
        t = t + 1

        # Accuracy computation of combinations of features
        Counter = 0
        counting = 0
        for counting in range(np.size(predicting)):
            if testB[counting][0] == predicting[counting]:
                Counter = Counter + 1
            counting = counting + 1
        acc=(Counter / counting) * 100
        print("Accuracy for subset: ",newData.columns,"is: ", acc)
        combination_accuracies.append(acc)

    #selecting the best combination:
    best_combo_value=max(combination_accuracies)
    best_combo_index=combination_accuracies.index(best_combo_value)+1
    #print(best_combo_index)

    if(best_combo_value>best_feature_value): #check if the accuracy of the model is
improved by the new subset
                                    # if yes, combine it further, if no, present it as the best
combination
        t = 1
        new_combination_accuracies = []
        for counter3 in range(np.size(allFeatures) - 1):

            newData = data2.iloc[:, [0, best_feature,best_combo_index, t]]
            #print(newData.columns)
            a = newData.iloc[:, :].values
            b = newData.iloc[:, :].values
            A, testA, B, testB = train_test_split(a, b, test_size=0.2, random_state=0)
            #print("A", A.shape)
            model.storeTrainingData(A, B)
```

```
            predicting = model.classify(testA, k=15)
            t = t + 1

            # Accuracy computation
            Counter = 0
            counting = 0
            for counting in range(np.size(predicting)):
                if testB[counting][0] == predicting[counting]:
                    Counter = Counter + 1
                counting = counting + 1
            acc = (Counter / counting) * 100
            new_combination_accuracies.append(acc)
            #print("Accuracy:",acc)

        new_best_combo_value = max(new_combination_accuracies)
        new_best_combo_index =
new_combination_accuracies.index(best_combo_value) + 1
        #print(new_best_combo_index)

        if(new_best_combo_value>best_combo_value): #check if the accuracy of the
model is improved by the new subset
                                    # if yes, combine it further, if no, present it as the best
combination
            t = 1
            new_combination_accuracies = []
            for counter4 in range(np.size(allFeatures) - 1):

                newData = data2.iloc[:, [0, best_feature,
best_combo_index,new_best_combo_index, t]]
                #print(newData.columns)
                a = newData.iloc[:, :].values
                b = newData.iloc[:, :].values
                A, testA, B, testB = train_test_split(a, b, test_size=0.2, random_state=0)
                #print("A", A.shape)
                model.storeTrainingData(A, B)
                predicting = model.classify(testA, k=15)
                t = t + 1

                # Accuracy computation
                Counter = 0
                counting = 0
                for counting in range(np.size(predicting)):
                    if testB[counting][0] == predicting[counting]:
                        Counter = Counter + 1
                    counting = counting + 1
                acc = (Counter / counting) * 100
```

```
            new_combination_accuracies.append(acc)
            #print("Accuracy:", acc)

        new_best_combo_value = max(new_combination_accuracies)
        new_best_combo_index =
new_combination_accuracies.index(best_combo_value) + 1
        #print(new_best_combo_index)

    else:
        print("The best feature subset is: Feature ", best_feature,", Feature:",
            best_combo_index, " With the accuracy: ", best_combo_value)


  else:
    print("The best feature subset is: Feature ",best_feature," With the accuracy:
",best_feature_value )


Backward Feature Selection :
  #Backward Elimination
  print("\nBackward Feature Elimination\n")

  Backward_accuracies=[]
  data3 = normalized_data.iloc[:, :]
  allFeatures = data3.columns
  i = np.size(allFeatures)
  # print(allFeatures)
  cols=[]

  for counter in range(np.size(allFeatures)-1):
    newData = data3.iloc[:, :i]
    cols.append(newData.columns)
    i = i - 1
    c = newData.iloc[:, :].values
    d = newData.iloc[:, :].values
    C, testC, D, testD = train_test_split(c, d, test_size=1 / 3, random_state=0)
    print("C", C.shape)
    model.storeTrainingData(C, D)
    predicting = model.classify(testC, k=14)
    # Accuracy computation
    Counter = 0
    counting = 0
    for counting in range(np.size(predicting)):
      if testC[counting][0] == predicting[counting]:
        Counter = Counter + 1
      counting = counting + 1
```

```
    accurate=(Counter / counting) * 100
    print("Back accuracy: ", accurate)
    Backward_accuracies.append(accurate)

backward_best=max(Backward_accuracies)
backward_index=Backward_accuracies.index(backward_best)
g=0
for h in range(backward_index):
    g=g+1


print("Best feature subset: ",cols[g], "\nWith Accuracy: ", backward_best)
```