# Project 2  CS205: Artificial Intelligence. Dr. Eamonn Keogh

**Rahul Sailesh Wadhwa**
**SID 862309846**
Email rwadh004@ucr.edu
Date: June 10, 2022

In completing this assignment I consulted:

- https://docs.python.org/3.6/contents.html
- The Machine Learning 001 and Machine Learning 002 lecture slides provided by Dr. Eamonn Keogh
- The assignment materials and sample report provided by Dr. Eamonn Keogh

All my code is original and implemented from scratch by me. I also made use of the following built-in subroutines-

- All subroutines from **numpy**, for manipulating the data as a ndarray to simplify my calculations.
- **sqrt** function from **math** to find the square root in the Euclidean distance calculation function.
- Subroutines from **matplotlib** to plot graphs to make the results more presentable.

Outline of this report:

# CS205: Project 2: Feature Selection with Nearest Neighbor
# Rahul Sailesh Wadhwa, SID 862309846 Date: June 10, 2022

It is a well-known fact that the nearest neighbor algorithm is a very good choice for classification tasks. It is a very simple yet effective classification algorithm. However one of the major drawbacks of this algorithm is that it is very sensitive to irrelevant features.

In this project, we are tasked with implementing feature selection along with the nearest neighbors algorithm. We are tasked with implementing 2 kinds of feature selection-

1. Forward selection - Forward feature selection is an iterative approach in which we start with having no features and then we iteratively add features to the dataset based on testing these features with the nearest neighbor algorithm. These features are then tested in conjunction with the existing list of features that obtain the highest accuracy.
2. Backward elimination - This is a feature selection method in which we start with all the features and then iteratively remove the features to try and obtain a higher accuracy until we are left with only a single feature.

I was assigned CS205_SP_2022_SMALLtestdata__37.txt as my small dataset and CS205_SP_2022_Largetestdata__18.txt as my large dataset.

In Figure 1 we can see the results of applying forward selection on the small dataset assigned to me.

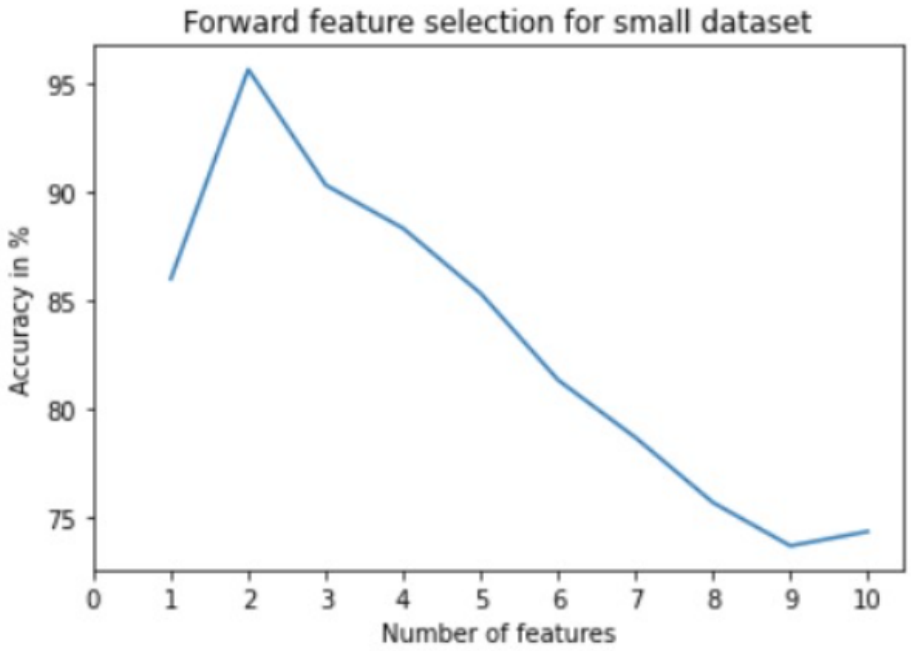


Figure 1. Plot of Accuracy vs Number of features for Forward Feature Selection on the small dataset.

At the beginning of the search, the feature “8” was chosen and it gave an accuracy of around 86%. Adding feature “2” further increased the accuracy of the model to around 95.6% and then adding feature “1” caused the accuracy of the model to drop to around 90.3%. Adding more features caused the accuracy of the model to decrease even further.

Figure 2 shows the results of running Backward elimination on the small dataset.

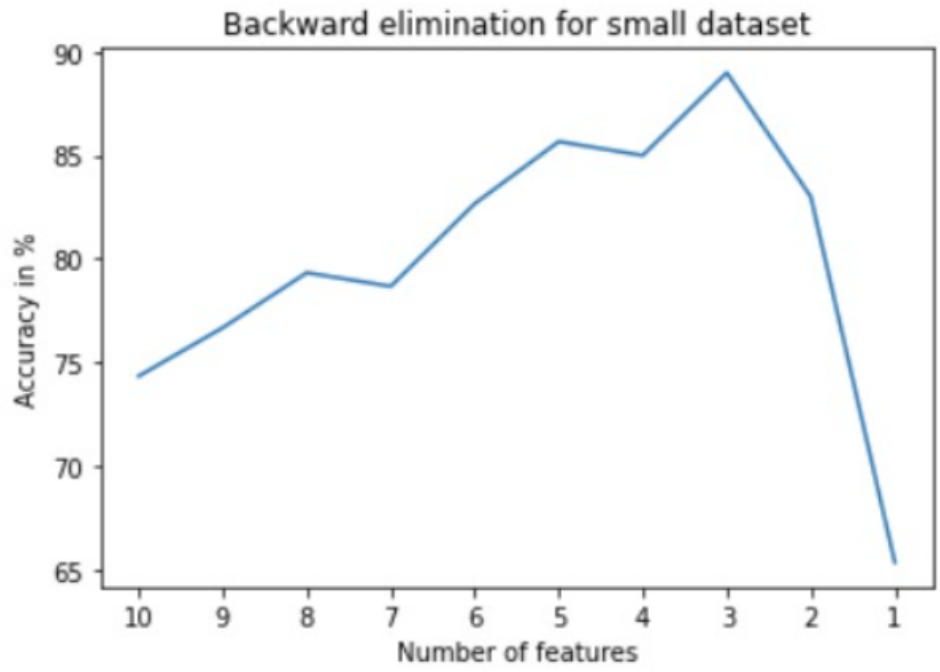


Figure 2. Plot of Accuracy vs Number of features for Backward Elimination on the small dataset.

First, for the sanity check, the accuracy of the last iteration of the forward selection should match the first iteration of the backward elimination algorithm. At the beginning of the search when taking all of the features into consideration, an accuracy of around 74.3% was obtained. On eliminating the different features a peak accuracy of around 89% was obtained when taking features “2”,”8”, and “10” into consideration.

**Conclusion for the small dataset:**
There is reason to believe that features “8” and “2” can be mostly used to classify the data with very high accuracy and there is also evidence that the feature “1” and “10” may also be useful, however, this has to be evaluated further before making any conclusion. If we deploy the {“8”,”2”} model, I believe that we will be able to obtain an accuracy of around 95.6%.

Now for the larger dataset, it contained around 1000 rows and 40 features as opposed to the 300 rows and 10 features of the smaller dataset.

In Figure 3 we can see the results of applying the forward feature selection method to the larger dataset.

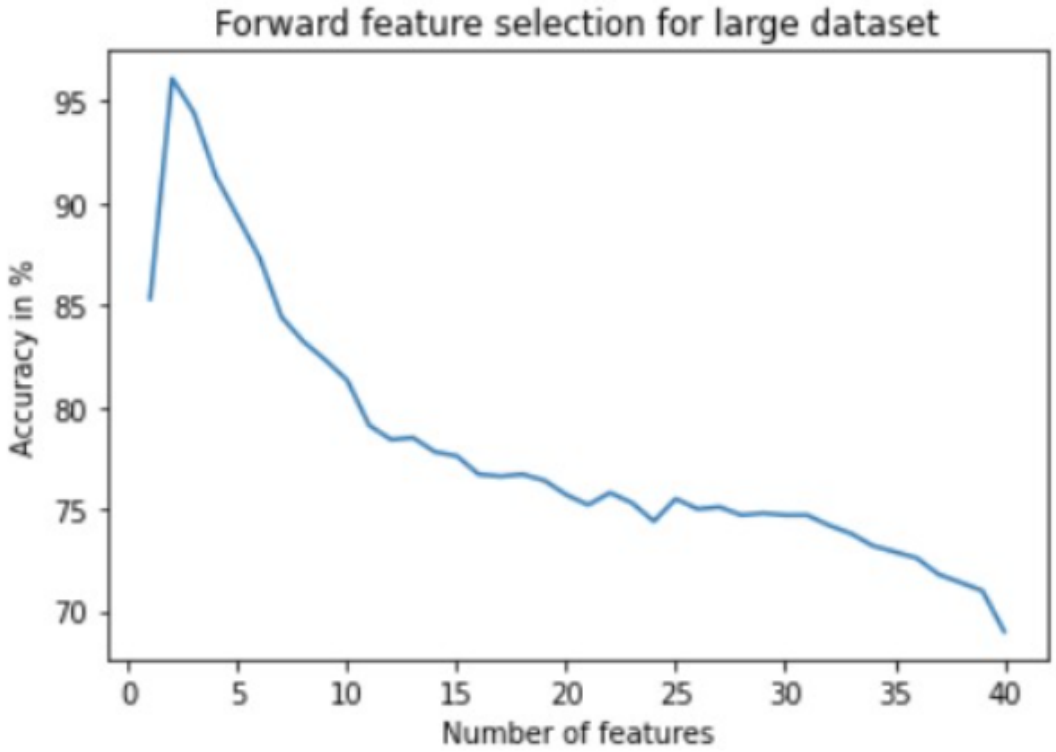


Figure 3. Plot of Accuracy vs Number of features for Forward Feature Selection on the large dataset

At the beginning of the search, the feature “18” was chosen and it gave an accuracy of around 85.3%. Adding feature “19” further increased the accuracy of the model to around 96.1% and then adding feature “11” caused the accuracy of the model to decrease giving us an accuracy of around 94.4%. Adding more features caused the accuracy of the model to decrease even further.

Figure 4 shows the result of applying Backward Elimination to the large dataset.

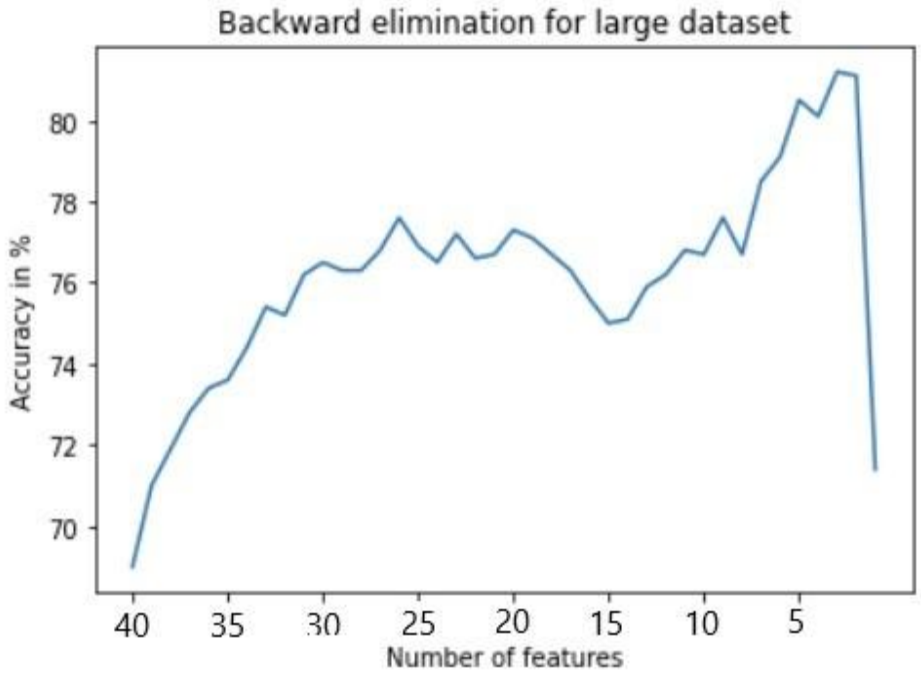


Figure 4. Plot of Accuracy vs Number of features for Backward Elimination on the large dataset

First, for the sanity check, the accuracy of the last iteration of the forward selection should match the first iteration of the backward elimination algorithm. At the beginning of the search when taking all of the features into consideration, an accuracy of around 69% was obtained. On eliminating the different features a peak accuracy of around 81.2% was obtained when taking features “6”,”18”, and “40” into consideration.

**Conclusion for the large dataset:**
There is reason to believe that features “18” and “19” can be mostly used to classify the data with very high accuracy. There is also evidence that the features “11”,”6” and “40” may also be useful, however, this has to be evaluated further before making any conclusion. If we deploy the {“18”,”19”} model, I believe that we will be able to obtain an accuracy of around 96.1%.

**Computational effort for search:**
I implemented the search in Jupyter Notebook, and ran all experiments on a laptop with a Intel Core i7-1165G7 CPU and 16 gigs of RAM. In the table shown below we can observe the various times taken to run my code for the different algorithms.

| Algorithm | Small Dataset | Large Dataset |
|---|---|---|
| Forward Feature Selection | 10.5s | 1.5 hours |
| Backward Elimination | 12.8 | 2 hours |

**Below is a trace of my algorithm for Forward Feature Selection on the small dataset.**

Feature Selection algortihm

Enter the name of the file to test: small_37.txt
Type the number of the algortihm you want to run.
1. Forward Selection
2. Backward Elimination
1
This dataset has 10 features and 300 instances.

At tree depth 1
Considering adding feature(s) (0)
Gave an accuracy of 65.33%
Considering adding feature(s) (1)
Gave an accuracy of 72.0%
Considering adding feature(s) (2)
Gave an accuracy of 68.67%
Considering adding feature(s) (3)
Gave an accuracy of 67.0%
Considering adding feature(s) (4)
Gave an accuracy of 65.67%
Considering adding feature(s) (5)
Gave an accuracy of 67.33%
Considering adding feature(s) (6)
Gave an accuracy of 65.67%
Considering adding feature(s) (7)
Gave an accuracy of 67.67%
Considering adding feature(s) (8)
Gave an accuracy of 86.0%
Considering adding feature(s) (9)
Gave an accuracy of 62.67%

Adding feature (8)
Considering the feature(s): ([8])
Gave best accuracy of 86.0%
…
//// Here I deleted most of the trace to save space. However i have shown the first and last levels of search///
…
Adding feature (3)
Considering the feature(s): ([8, 2, 1, 9, 5, 4, 0, 6, 7, 3])
Gave best accuracy of 74.33%

Best features after running the search: ([8, 2])

Gave an accuracy of 95.67%
## URL to my code is:
https://drive.google.com/file/d/1U2TIgFzDIA-MoowHYLWSIxyGtUOhPX-_/view?usp=sharing

# CODE

In [1]:
```
import numpy as np
from math import sqrt
import matplotlib.pyplot as plt
```

In [2]:
```
#setting the small and large datasets
sfile='small_37.txt'
lfile='large_18.txt'
```

In [3]:
```
#function to load the data
def load_data(filename):
    data, labels = [], []

    with open(filename) as f:
        lines = f.readlines()

    for item in lines:
        temp=item.split()
        temp=np.float_(temp)
        labels.append(temp[0])
        temp=np.delete(temp,0)
        data.append(temp)

    data=np.array(data)
    labels=np.array(labels)
    return data,labels
```

In [6]:
```
#function for leave one out cross validation when using backward elimination
def cross_valid_backward(data,labels,features,testfeature):
    #creating a copy of the feature list
    tempo=features.copy()
    #for the case when all features have to be considered
    if testfeature==-1:
        pass
    #removing the feature being tested from the list of features
    else:
        tempo.remove(testfeature)

    ctr=0

    #finding the nearest neighbor with the least euclidean distance
    for i in range(len(data)):
        nearest_neighbor = np.inf
        nearest_neighbor_index=0
        for j in range(len(data)):
            dist=euclid(data[i],data[j],tempo)
            if(nearest_neighbor > dist and j != i):
                nearest_neighbor=dist
                nearest_neighbor_index = j

        #incrementing the counter if the predicted label(using the nearest neighbor) matches the actual label
        if(labels[i]==labels[nearest_neighbor_index]):
            ctr+=1

    #returning the accuracy
    return ctr/len(data)
```

In [7]:

```
#function to perform backward elimination for feature selection
def backward_feature_selection(data,labels):
    #initializing the feature list ot contain all the features
    cur_features=list(range(1,len(data[0])+1))
    highestaccu=0
    res_features=[]
    accu_list=[]
    depth=0

    #calculating the accuracy with all the features
    accu=cross_valid_backward(data,labels,cur_features,-1)
    print("At tree depth {0}".format(depth))
    print("With all the features i.e. ({0}) accuracy is {1}".format(cur_features,round(accu*100,2)))
    accu_list.append(round(accu*100,2))

    #feature elimination loop
    for i in range(len(data[0])):
        testfeature=-1
        bestaccu=0
        depth+=1

        #if only one feature is present in the list of features being considered then stop
        if(len(cur_features)==1):
            break


        print("\n\nAt tree depth {0}".format(depth))

        for j in range(len(data[i]-1)):
            if (check_if_already_visited(cur_features,j)==1):

                accu=cross_valid_backward(data,labels,cur_features,j)
                tempo=cur_features.copy()
                tempo.remove(j)
```

In [8]:

```
#function for leave one out cross validation when using forward selection
def cross_valid_forward(data,labels,features,testfeature):

    #Adding the current feature being tested to the list of features
    temp=features+[testfeature]
    ctr=0

    #finding the nearest neighbor
    for i in range(len(data)):
        nearest_neighbor = np.inf
        nearest_neighbor_index=0
        for j in range(len(data)):
            dist=euclid(data[i],data[j],temp)
            if(nearest_neighbor > dist and j != i):
                nearest_neighbor=dist
                nearest_neighbor_index = j

        #incrementing the counter if the predicted label(using the nearest neighbor) matches the actual label
        if(labels[i]==labels[nearest_neighbor_index]):
            ctr+=1

    #returning the accuracy
    return ctr/len(data)
```

In [9]:

```
#function to perform forward feature selection
def forward_feature_selection(data,labels):
    cur_features=[]
    highestaccu=0
    res_features=[]
    accu_list=[]
    depth=0

    #forward feature selection loop
    for i in range(len(data[0])):
        #initialization
        testfeature=-1
        bestaccu=0
        depth+=1

        print("\n\nAt tree depth {0}".format(depth))

        for j in range(len(data[i])):

            #checking whether the feature is not already in the feature list
            if (check_if_already_visited(cur_features,j)==0):

                #calculating the accuracy
                accu=cross_valid_forward(data,labels,cur_features,j)

                #for single features
                if not cur_features:
                    print("Considering adding feature(s) ({0}) \nGave an accuracy of {1}%".format(j,round(accu*100,2)))

                #when considering multiple features
                else:
                    print("Considering adding feature ({0})\nAnd testing using existing feature(s) ({1}) \nGave an accura

                #updating the best accuracy and deciding which feature to add
                if(accu>bestaccu):
                    bestaccu=accu
```

In [ ]:

```
#wrapper function to call the differnt methods
def wrapper():
    print("Feature Selection algortihm")
    filename=input("\nEnter the name of the file to test: ")
    ch=int(input("\nType the number of the algortihm you want to run.\n1. Forward Selection\n2. Backward Elimination\n"))
    data,labels=load_data(filename)
    print("\nThis dataset has {0} features and {1} instances.".format(data.shape[1],data.shape[0]))
    if(ch==1):
        best_accu=forward_feature_selection(data,labels)
        forward_plotter(best_accu)
    else:
        best_accu=backward_feature_selection(data,labels)
        backward_plotter(best_accu)
wrapper()
```

In [ ]:

```
def forward_plotter(best_accu):
    #plotting graph for forward selection
    plt.plot(range(1,len(data[0])+1,1),best_accu)
    plt.xlabel("Number of features")
    plt.ylabel("Accuracy in %")
    plt.title('Forward feature selection for large dataset')
```

In [ ]:

```
def backward_plotter(best_accu):
    #plotting the graph for backward elimination
    plt.plot(range(1,len(data[0])+1),best_accu)
    plt.xlabel("Number of features")
    x = np.arange(1, len(data[0])+1, 5)
    plt.xticks(x,x[::-1])
    plt.ylabel("Accuracy in %")
    plt.title('Backward elimination for large dataset')
```