

# CS205, Artificial Intelligence, Dr. Eamonn Keogh

## Feature Selection with Nearest Neighbor

Md Rayhanul Masud (SID: 862317259)  
Email mmasu012@ucr.edu

Jun 10 2022

**References** The references that I have followed to complete the assignment

- The lecture contents related to the Nearest Neighbor, Cross Validation and Feature Selection topics
- Documentation of Python 3.10 <https://docs.python.org/3>
- The project description pdf provided by the professor
- The sample report for this project provided by the professor

**Code Implementation** The python code implementation of this project is solely original except the library/packages from python

- **pandas** to read the dataset files for the experimentation of the project
- **time** to calculate the execution time needed to run the code
- **matplotlib** to generate plots

and the slide Project\_2\_Briefing.pptx provided by the professor

### Outline of the report

- Cover Page (Page 1)
- The description of Nearest neighbor classifier (Page 2)
- The description how forward selection and backward elimination work (Page 2)
- Analysis of the Experimentation details and results with plots (Page 3)
- Conclusion (Page 4)
- The sample trace of a run on the small dataset provided with Forward Selection Method (Page 5)
- The source code is attached at the end of this report, and also upload in <https://github.com/mmasu012/Feature-Selection-with-Nearest-Neighbor>

# 1 Introduction

## 1.1 What is Nearest Neighbor Algorithm

Nearest neighbor algorithm is a simple algorithm which assigns the class of an instance based on the majority vote of its closest neighbors. The distance between the instances can be defined by any distance measurement (i.e. euclidean distance). An instance can have a good number of features. When computing the distance across two instances, any subset of features can be considered. In the simplest form of nearest neighbor algorithm, the predicting instance is matched with the label of the closest neighbor, which can be denoted as 1-NN classification. In this project, this simplest form is carried out using the two assigned datasets by the professor.

## 1.2 What is Nearest Neighbor Algorithm

Feature selection is the way to find out the most significant features to carry out the task of prediction. It helps to immune the model from overfitting, and decrease the training time and increase the accuracy of the model removing the weakest features from the consideration. This project deals with the two most popular feature selection methods which are discussed later.

## 1.3 What is Forward Selection

Forward selection is a way of feature engineering to identify the strongest features that should be used for training a model of a machine learning based model. The method starts with an empty feature set. Then, in each step, it finds the best feature to be added to the feature set. For example: after the first step, if it finds that feature x is the best among all deriving the best accuracy, then the next step is to find which other feature along with feature x can provide the best accuracy having 2 features.

## 1.4 What is Backward Elimination

Backward elimination is a way of feature engineering to identify the strongest features that should be used for training a model of a machine learning based model. The method starts with all the features. Then, in each step, it finds the least significant feature to be removed from the feature set. For example: after the first step, if it finds that feature x is the least significant among all deriving the lowest accuracy, then the next step is to find which other feature is the least significant among the feature set (after removing feature x).

## 1.5 Goal of the project

This project is aimed to implement nearest neighbor algorithm and use it inside a wrapper of two feature selection procedures: forward selection and backward elimination to identify the most significant features to be found for the assigned datasets. The conclusion over the experiments is provided with the results found in respect of results and computational effectiveness in the later sections of this report.

# 2 Experiment and Results

## 2.1 Dataset Details

We have been assigned two datasets: among them one contains 10 features with 300 instances, and the larger one consists of 40 features with 1000 instances. Using **panda** library, the datasets were read in the program.

## 2.2 Experimentation Details

### 2.2.1 Selected features

Nearest neighbor algorithm is applied on the given datasets. In this project, to classify an instance, the closest neighbor is considered, which is 1 nearest neighbor. To validate the accuracy, we find that leave-one-out validation results in better performance as opposed to using 10-fold cross validation. The project guideline provides one large and one small dataset. It is found that in case of small dataset, if leave-one-out validation is used, both forward selection and backward elimination provide the same accuracy and select the same set of significant features. On the otherhand, 10-fold cross validation is used, though in case of forward selection, it selects the same feature set as the leave-one-out, the accuracy drops to 86%. Due to resource constraints, 10-fold cross validation is used for the large dataset. From the Table 1, it is evident that 10-fold cross validation for forward selection method shows accuracy near to 90%, backward elimination derives accuracy 80.8% with a quite bigger feature set. Both Figure 1 and Figure 2 show that in a huge

Feature Selection Method	Small Dataset (Used leave-one-out validation)	Small Dataset (Used 10-fold cross validation)	Large Dataset (Used 10-fold cross validation)
Forward Selection	{7,8}, 96.3%	{7,8}, 86.0%	{4,21}, 87.9%
Backward Elimination	{7,8}, 96.3%	{3,5,6,9,10}, 76.7%	{2,3,4,9,10,11,12,13,14,16,17,18,24,26,27,32,33,34,36,38,39}, 80.8%

Table 1: Selected most significant features with best validation accuracy for different experiments

feature space, most of the features are unnecessary and redundant. The more redundant features are used, the less accuracy can be achieved.

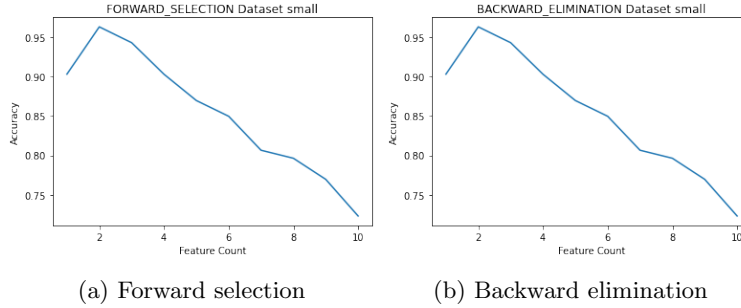


Figure 1: Accuracy vs Count of significant feature in the small dataset

## 2.3 Computational effort for search

All the experiments are carried out in MacBook Pro with 16 GB of RAM in the Jupyter notebook. Due to resource constraints, for small dataset, both 10-fold and leave-one-out cross validation are used, but in case of large dataset only 10-fold cross validation is used. Table 2 shows that for backward elimination, the execution time is quite larger than forward selection.

## 2.4 Usage of Cache

Since the execution time for the dataset containing large number of features gets increased exponentially, a cache is used to store the euclidean distance between two features. It reduces the execution time to a great extent.

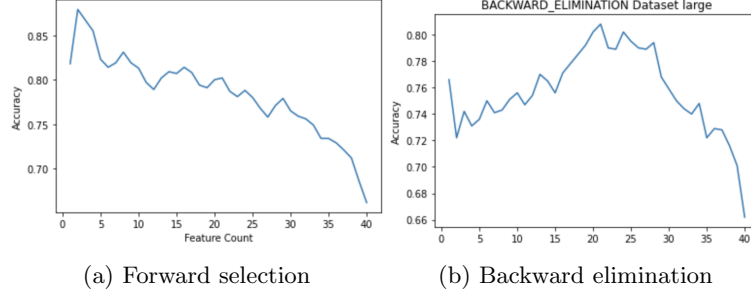


Figure 2: Accuracy vs Count of significant feature in the large dataset

Feature Selection Method	Small Dataset (Used leave-one-out validation)	Small Dataset (Used 10-fold cross validation)	Large Dataset (Used 10-fold cross validation)
Forward Selection	1.4 hrs	0.8 sec	3.6 mins
Backward Elimination	2.25 hrs	1.5 sec	6.5 mins

Table 2: Execution time of the experiments

## 2.5 Instructions to run the source code

The source is uploaded <https://github.com/mmasu012/Feature-Selection-with-Nearest-Neighbor>. There is a jupyter notebook which can be run to test the implementation. The code is also added to the end of this report.

## 3 Conclusion

- Feature selection is essential when the dataset contains a large number of features. It is important to identify the most significant features from the pool.
- Forward selection and backward elimination may not derive the same result, since the methods start with empty set and the set with all features respectively.
- In the experiment, we find that forward selection works better than backward elimination.
- For large number of features, 10-fold cross validation is more reasonable than using leave-one-out cross validation. The execution time increases exponentially with the number of features in the dataset.
- For nearest neighbor algorithm, if the feature set is small, forward selection and backward elimination can play significant roles in the identification of strongest features to train the model and can result in better accuracy.

## 4 Sample Trace of running the project

```
Press 11 to run Forward Selection with small dataset
Press 12 to run Forward Selection with large dataset
Press 21 to run Backward Elimination with small dataset
Press 22 to run Backward Elimination with large dataset
Press any other key to exit
11
Forward Selection selected with small dataset

FORWARD_SELECTION
Running nearest neighbor with all 10 features, using leave-one-out evaluation with accuracy 72.3%

Beginning Search
On the 1 th level of the search tree
  Using features {1} accuracy is 67.4%
  Using features {2} accuracy is 65.7%
  Using features {3} accuracy is 69.7%
  Using features {4} accuracy is 66.3%
  Using features {5} accuracy is 71.0%
  Using features {6} accuracy is 73.3%
  Using features {7} accuracy is 76.3%
  Using features {8} accuracy is 90.3%
  Using features {9} accuracy is 69.7%
  Using features {10} accuracy is 68.0%
Feature set {8} was best. accuracy is 90.3% adding feature 8

On the 2 th level of the search tree
  Using features {1,8} accuracy is 88.0%
  Using features {2,8} accuracy is 82.7%
  Using features {3,8} accuracy is 83.0%
  Using features {4,8} accuracy is 81.3%
  Using features {5,8} accuracy is 81.7%
  Using features {6,8} accuracy is 84.7%
  Using features {7,8} accuracy is 96.3%
  Using features {8,9} accuracy is 82.0%
  Using features {8,10} accuracy is 85.0%
Feature set {7,8} was best. accuracy is 96.3% adding feature 7

On the 3 th level of the search tree
  Using features {1,7,8} accuracy is 89.0%
  Using features {2,7,8} accuracy is 89.3%
  Using features {3,7,8} accuracy is 90.0%
  Using features {4,7,8} accuracy is 87.3%
  Using features {5,7,8} accuracy is 91.7%
  Using features {6,7,8} accuracy is 92.0%
  Using features {7,8,9} accuracy is 93.0%
  Using features {7,8,10} accuracy is 94.3%
Feature set {7,8,10} was best. accuracy is 94.3% adding feature 10

On the 4 th level of the search tree
  Using features {1,7,8,10} accuracy is 85.7%
  Using features {2,7,8,10} accuracy is 85.3%
  Using features {3,7,8,10} accuracy is 87.7%
  Using features {4,7,8,10} accuracy is 89.6%
  Using features {5,7,8,10} accuracy is 85.7%
  Using features {6,7,8,10} accuracy is 90.3%
  Using features {7,8,9,10} accuracy is 87.0%
Feature set {6,7,8,10} was best. accuracy is 90.3% adding feature 6

On the 5 th level of the search tree
  Using features {1,6,7,8,10} accuracy is 85.6%
  Using features {2,6,7,8,10} accuracy is 81.3%
  Using features {3,6,7,8,10} accuracy is 83.0%
  Using features {4,6,7,8,10} accuracy is 87.0%
  Using features {5,6,7,8,10} accuracy is 84.3%
  Using features {6,7,8,9,10} accuracy is 83.6%
Feature set {4,6,7,8,10} was best. accuracy is 87.0% adding feature 4

On the 6 th level of the search tree
  Using features {1,4,6,7,8,10} accuracy is 79.7%
  Using features {2,4,6,7,8,10} accuracy is 81.6%
  Using features {3,4,6,7,8,10} accuracy is 85.0%
  Using features {4,5,6,7,8,10} accuracy is 79.3%
  Using features {4,6,7,8,9,10} accuracy is 79.0%
Feature set {3,4,6,7,8,10} was best. accuracy is 85.0% adding feature 3

On the 7 th level of the search tree
  Using features {1,3,4,6,7,8,10} accuracy is 78.0%
  Using features {2,3,4,6,7,8,10} accuracy is 78.3%
  Using features {3,4,5,6,7,8,10} accuracy is 80.7%
  Using features {3,4,6,7,8,9,10} accuracy is 74.0%
Feature set {3,4,5,6,7,8,10} was best. accuracy is 80.7% adding feature 5

On the 8 th level of the search tree
  Using features {1,3,4,5,6,7,8,10} accuracy is 79.6%
  Using features {2,3,4,5,6,7,8,10} accuracy is 77.7%
  Using features {3,4,5,6,7,8,9,10} accuracy is 78.0%
Feature set {1,3,4,5,6,7,8,10} was best. accuracy is 79.6% adding feature 1

On the 9 th level of the search tree
  Using features {1,2,3,4,5,6,7,8,10} accuracy is 76.3%
  Using features {1,3,4,5,6,7,8,9,10} accuracy is 77.0%
Feature set {1,3,4,5,6,7,8,9,10} was best. accuracy is 77.0% adding feature 9

Finished search !! The best feature subset is {7,8} which has an accuracy of 96.3%
```



```
import pandas as pd
import math
import time
```

```
In [ ]: # feature selection type variables
FORWARD_SELECTION, BACKWARD_ELIMINATION = 'forward_selection', 'backward_elimination'

# cache to reduce the computational cost during measurement of euclidean distance
cache = {}
```

```
In [ ]: # given a file path of the dataset, method reads dataset
# and returns all the instances with their labels
# and feature count per instance
def read_dataset(dataset_path):

    data = pd.read_csv(dataset_path, delim_whitespace=True, header = None)
    instance_count, column_count = data.shape
    feature_count = column_count - 1
    instances = data.values.tolist()

    return instances, instance_count, feature_count
```

```
In [ ]: # given two instances of the dataset, and a list of features to be considered
# method find the euclidean distance based on the features given
# the method uses a global cache to reduce the computational cost incurred
# to find the distance, whenever it finds two features, that are already stored
# in the map with its squared difference, the method gets the value from the map
# otherwise, it calculates the value and stores in the map for further usage
def find_euclidean_distance(instance, compare_instance, features):

    squares = 0
    # iterates over the given features
    for feature in features:

        x = instance[feature]
        y = compare_instance[feature]
        val = 0

        # searching if already the calculated value is in the cache
        if (x,y) in cache:
            val = cache[(x, y)]
        elif (y,x) in cache:
            val = cache[(y, x)]
        else:
            diff = x - y
            val = diff ** 2

            cache[(x, y)] = val

        squares += val

    return math.sqrt(squares)
```

```
In [ ]: # given the dataset and feature set to be considered
# method predicts the label of the instance and compares the predicted one with
# the given label using nearest neighbor classifier where k = 1
# method returns the ratio of count of correctly predicted instances and
# total instances
def find_nearest_neighbor(dataset, feature_set, instance_count):

    correct_prediction = 0
    # iterating over each of the instance
    for instance_idx in range(instance_count):

        # extracts the current instance
        instance = dataset[instance_idx]
        # extracts the label of the current instance
        target = instance[0]
        # the set of features to be considered
        features = feature_set

        nearest_neighbor_distance = math.inf
        nearest_neighbor_predict = -1

        # iterates over each of the other instances in the dataset
        # to find which instance is more close in respect of euclidean
        # distance
        for compare_idx in range(instance_count):

            if compare_idx != instance_idx:

                compare_instance = dataset[compare_idx]
                compare_target = compare_instance[0]
                distance = find_euclidean_distance(instance, compare_instance, features)

                if distance < nearest_neighbor_distance:
                    nearest_neighbor_distance = distance
                    nearest_neighbor_predict = compare_target

        # checking whether the predicted label matches the target label
        if nearest_neighbor_predict == target:
            correct_prediction += 1

    # returns the ratio of the count of correctly predicted instance and
    # total instances
    return correct_prediction / instance_count
```

```
In [ ]: import matplotlib.pyplot as plt

# plots a graph to show how accuracy gets affected by feature counts
def plot_graph(accuracy_map, search_type, dataset_size):

    x = []
    y = []
    for key in sorted(accuracy_map):
        x.append(key)
        y.append(accuracy_map[key])

    plt.plot(x, y)
    plt.xlabel('Feature Count')
    plt.ylabel('Accuracy')
    plt.title(' ' + search_type.upper() + ' Dataset ' + dataset_size)
    plt.show()
```

```
In [ ]: # given the instances and feature set to be considered,
# method iteratively leaves one instance, runs nearest neighbor classifier on
# the remaining instances, and finds the accuracy, that is how, the number of
# validation set will be equal to the number of instance count
# returns the average of the accuracy over all the validation sets
def leave_one_out_cross_validation(instances, instance_count, feature_set):

    accuracy_list = []
    # iterates over each of the instances
    for k_fold_itr in range(0, instance_count):

        # prepares a dataset removing k_fold_itr th instance
        dataset = instances[ :k_fold_itr] + instances[k_fold_itr + 1:]

        # runs nearest neighbor classifier to finds accuracy
        accuracy = find_nearest_neighbor(dataset, feature_set, \
                                         instance_count - 1)

        # appends the accuracy in a list
        accuracy_list.append(accuracy)

    # returns the average accuracy found
    return sum(accuracy_list)/instance_count
```

```
In [ ]: # given the type of feature selection type along with all the instances
# selection given)
# iterates over different sets of features (based on the type of feature
# selection)
# for each different set of feature, finds leave-one-out validation accuracy
# using nearest neighbor classifier (k = 1)
# prints the best accuracy with the best feature set
# returns the best accuracy to best feature set per level to plot graph
def start_experiment(instances, instance_count, feature_count, \
                    search_type = FORWARD_SELECTION):

    print()
    print()
    print(search_type.upper())
    accuracy_map = {}

    all_feature_set = list(range(1, feature_count + 1))
    accuracy = leave_one_out_cross_validation(\
        instances, instance_count, all_feature_set)
    print('Running nearest neighbor with all', feature_count, \
        'features, using leave-one-out evaluation',
        'with accuracy', '{:0.1f}%'.format(accuracy * 100))
    accuracy_map[len(all_feature_set)] = accuracy

    # starts with an empty feature set for forward selection
    # starts with all features for backward elimination
    if search_type == FORWARD_SELECTION:
        current_feature_set = []
    elif search_type == BACKWARD_ELIMINATION:
        current_feature_set = all_feature_set

    print()
    print('Beginning Search ')
    best_accuracy_feature_set = all_feature_set
    best_accuracy = accuracy

    # iterating over each of the level
    # on each level, either the count of features
    # equal to (level count) (forward selection)
    # or (total feature count - level count) (backward elimination)
    for level in range(1, feature_count + 1):

        print('On the ', str(level), 'th level of the search tree')
        level_wise_best_accuracy_feature_set = None
        level_wise_best_accuracy = None
        level_wise_best_accuracy = 0

        # iterating over each of the features which can be added
        # or removed for feature selection/backward elimination
        # respectively
        for feature in range(1, feature_count + 1):

            if (feature not in current_feature_set and \
                search_type == FORWARD_SELECTION) or \
                (feature in current_feature_set and \
                 search_type == BACKWARD_ELIMINATION):

                if search_type == FORWARD_SELECTION:
                    feature_set = current_feature_set + [feature]
                elif search_type == BACKWARD_ELIMINATION:
                    feature_set = list(set(current_feature_set) - set([feature]))

                # finding leave one out cross validation accuracy
                accuracy = leave_one_out_cross_validation(\
                    instances, instance_count, \
                    feature_set)

                # sorting for displaying results in good manner
                feature_set.sort()
                feature_set_string = '(' + ','.join(str(f) \
                                                    for f in feature_set) + ')')
                print('\tUsing features', feature_set_string, 'accuracy is', \
                    '{:0.1f}%'.format(accuracy * 100))

                # checks if current feature set provides the better accuracy
                # among the feature sets in the current level
                if accuracy > level_wise_best_accuracy:
                    level_wise_best_accuracy = accuracy
                    level_wise_best_accuracy_feature_set = feature_set

            if search_type == FORWARD_SELECTION:
                current_feature_set.append(level_wise_best_accuracy_feature)
                add_remove_log = 'adding ' + 'Feature ' + \
                    str(level_wise_best_accuracy_feature)
            elif search_type == BACKWARD_ELIMINATION:
                current_feature_set = list(set(current_feature_set) - \
                                            set([level_wise_best_accuracy_feature]))
                add_remove_log = 'removing ' + 'Feature ' + \
                    str(level_wise_best_accuracy_feature)

            level_wise_best_feature_set_string = '(' + ','.join(str(f) \
                                                            for f in level_wise_best_accuracy_feature_set) + ')')
            print('Feature set ', level_wise_best_feature_set_string, \
                ' was best. accuracy is', '{:0.1f}%'.format(\
                    level_wise_best_accuracy*100), add_remove_log )

        accuracy_map[len(level_wise_best_accuracy_feature_set)] = \
            level_wise_best_accuracy
        print()

    # checks if current level accuracy is best among the so far
    # feature sets considered
    if level_wise_best_accuracy > best_accuracy:
        best_accuracy = level_wise_best_accuracy
        best_accuracy_feature_set = level_wise_best_accuracy_feature_set

    # displays the best accuracy calculated with the best feature set
    best_accuracy_feature_set_string = '(' + ','.join(str(f) \
                                                    for f in best_accuracy_feature_set) + ')')
    print('Finished search !!', 'The best feature subset is', \
        best_accuracy_feature_set_string, \
        'which has an accuracy of', '{:0.1f}%'.format(best_accuracy*100) )

    return accuracy_map
```

```
In [ ]: # driver function
if __name__ == "__main__":

    # small_dataset_path = 'CS205_CalibrationData_1.txt'

    small_dataset_path = 'CS205_SP_2022_SMALLtestdata_35.txt'
    large_dataset_path = 'CS205_SP_2022_Largestestdata_62.txt'

    while True:

        input_case = input('Press 11 to run Forward Selection with small dataset\n' + \
                            'Press 12 to run Forward Selection with large dataset\n' + \
                            'Press 21 to run Backward Elimination with small dataset\n' + \
                            'Press 22 to run Backward Elimination with large dataset\n' + \
                            'Press any other key to exit\n').strip()

        start_time = time.time()

        if input_case == '11':
            print('Forward Selection selected with small dataset')
            instances, instance_count, feature_count = read_dataset(small_dataset_path)
            accuracy_map = start_experiment(instances, instance_count, \
                                           feature_count, FORWARD_SELECTION)
            plot_graph(accuracy_map, FORWARD_SELECTION, 'small')

        elif input_case == '12':
            print('Forward Selection selected with large dataset')
            instances, instance_count, feature_count = read_dataset(large_dataset_path)
            accuracy_map = start_experiment(instances, instance_count, \
                                           feature_count, FORWARD_SELECTION)
            plot_graph(accuracy_map, FORWARD_SELECTION, 'large')

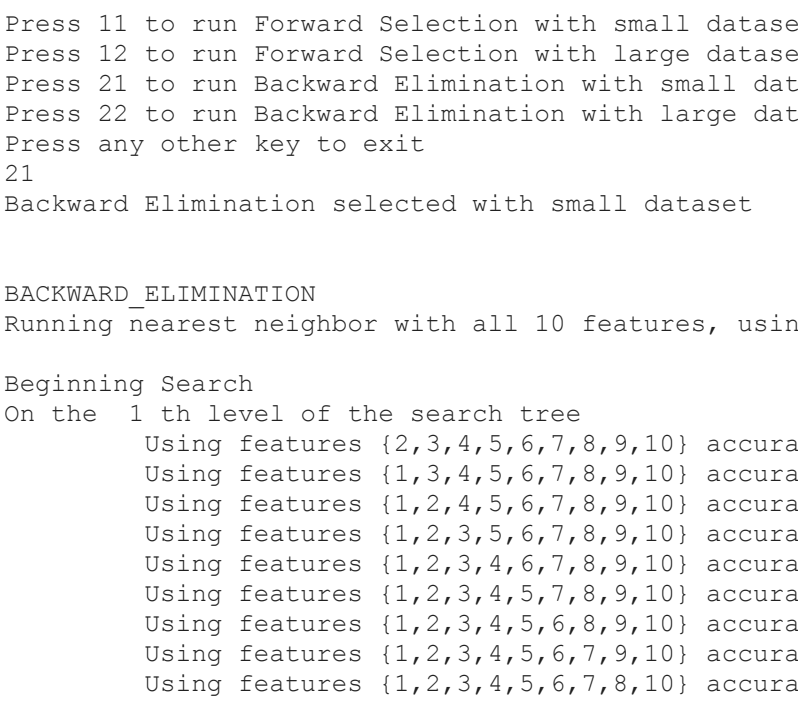
        elif input_case == '21':
            print('Backward Elimination selected with small dataset')
            instances, instance_count, feature_count = read_dataset(small_dataset_path)
            accuracy_map = start_experiment(instances, instance_count, \
                                           feature_count, BACKWARD_ELIMINATION)
            plot_graph(accuracy_map, BACKWARD_ELIMINATION, 'small')

        elif input_case == '22':
            print('Backward Elimination selected with large dataset')
            instances, instance_count, feature_count = read_dataset(large_dataset_path)
            accuracy_map = start_experiment(instances, instance_count, \
                                           feature_count, BACKWARD_ELIMINATION)
            plot_graph(accuracy_map, BACKWARD_ELIMINATION, 'large')

        else:
            print('Exit. Thank you')
            break

    end_time = time.time()
    execution_time_in_seconds = end_time - start_time
    print('Execution time ', '{:0.1f} seconds'.format(execution_time_in_seconds))

    print()
    print()
```



Execution time 4987.5 seconds

Press 11 to run Forward Selection with small dataset  
Press 12 to run Forward Selection with large dataset  
Press 21 to run Backward Elimination with small dataset  
Press 22 to run Backward Elimination with large dataset  
Press any other key to exit  
21  
Backward Elimination selected with small dataset

BACKWARD ELIMINATION  
Running nearest neighbor with all 10 features, using leave-one-out evaluation with accuracy 72.3%

Beginning Search  
On the 1 th level of the search tree  
Using features {1,3,4,5,6,7,8,9,10} accuracy is 75.0%  
Using features {2,3,4,5,6,7,8,9,10} accuracy is 77.0%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 74.0%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 73.3%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 69.0%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 72.3%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 71.0%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 83.6%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 76.3%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 72.7%  
Using features {1,2,3,4,5,6,7,8,9,10} accuracy is 76.3%  
Feature set {1,2,3,4,5,6,7,8,9,10} was best. accuracy is 77.0% removing feature 2

On the 2 th level of the search tree  
Using features {3,4,5,6,7,8,9,10} accuracy is 78.0%  
Using features {1,4,5,6,7,8,9,10} accuracy is 74.0%  
Using features {1,3,5,6,7,8,9,10} accuracy is 74.3%  
Using features {1,3,4,6,7,8,9,10} accuracy is 73.0%  
Using features {1,2,3,5,6,7,8,9,10} accuracy is 77.6%  
Using features {1,3,4,5,6,7,8,9,10} accuracy is 78.0%  
Using features {1,3,4,5,6,7,9,10} accuracy is 74.0%  
Using features {1,3,4,5,6,7,8,10} accuracy is 79.6%  
Using features {1,3,4,5,6,7,8,9,10} accuracy is 76.7%  
Feature set {1,3,4,5,6,7,8,10} was best. accuracy is 79.6% removing feature 9

On the 3 th level of the search tree  
Using features {3,4,5,6,7,8,10} accuracy is 80.7%  
Using features {1,4,5,6,7,8,10} accuracy is 79.3%  
Using features {1,3,5,6,7,8,10} accuracy is 78.3%  
Using features {1,3,4,5,6,7,8,10} accuracy is 78.0%  
Using features {1,3,4,5,6,7,8,10} accuracy is 79.3%  
Using features {1,3,4,5,6,7,10} accuracy is 75.0%  
Using features {1,3,4,5,6,7,8} accuracy is 76.0%  
Feature set {3,4,5,6,7,8,10} was best. accuracy is 80.7% removing feature 1

On the 4 th level of the search tree  
Using features {4,5,6,7,8,10} accuracy is 79.3%  
Using features {3,5,6,7,8,10} accuracy is 79.3%  
Using features {3,4,6,7,8,10} accuracy is 85.0%  
Using features {3,4,5,7,8,10} accuracy is 80.6%  
Using features {3,4,5,6,8,10} accuracy is 80.0%  
Using features {3,4,5,6,7,10} accuracy is 75.3%  
Using features {3,4,5,6,7,8} accuracy is 82.6%  
Feature set {3,4,6,7,8,10} was best. accuracy is 85.0% removing feature 5

On the 5 th level of the search tree  
Using features {4,6,7,8,10} accuracy is 87.0%  
Using features {3,6,7,8,10} accuracy is 83.0%  
Using features {3,4,7,8,10} accuracy is 81.7%  
Using features {3,4,6,8,10} accuracy is 83.6%  
Using features {3,4,6,7,10} accuracy is 72.7%  
Using features {3,4,6,7,8} accuracy is 86.0%  
Feature set {4,6,7,8,10} was best. accuracy is 80.0% removing feature 3

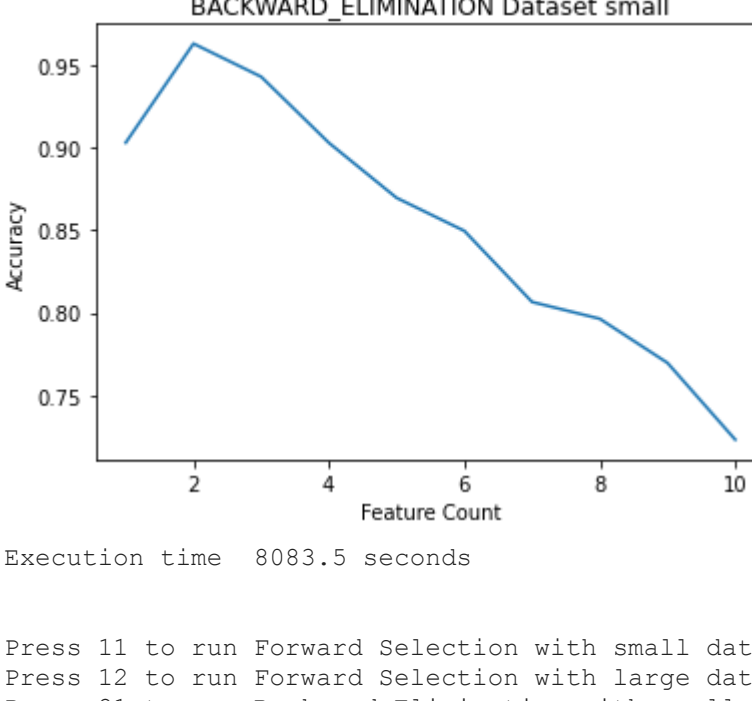
On the 6 th level of the search tree  
Using features {6,7,8,10} accuracy is 90.3%  
Using features {4,7,8,10} accuracy is 89.6%  
Using features {4,6,8,10} accuracy is 81.0%  
Using features {4,6,7,10} accuracy is 74.3%  
Using features {4,6,7,8} accuracy is 85.6%  
Feature set {6,7,8,10} was best. accuracy is 90.3% removing feature 4

On the 7 th level of the search tree  
Using features {7,8,10} accuracy is 94.3%  
Using features {6,8,10} accuracy is 84.7%  
Using features {6,7,10} accuracy is 74.3%  
Using features {6,7,8} accuracy is 92.0%  
Feature set {7,8,10} was best. accuracy is 94.3% removing feature 6

On the 8 th level of the search tree  
Using features {8,10} accuracy is 85.0%  
Using features {7,10} accuracy is 72.3%  
Using features {7,8} accuracy is 96.3%  
Feature set {7,8} was best. accuracy is 96.3% removing feature 10

On the 9 th level of the search tree  
Using features {8} accuracy is 90.3%  
Using features {9} accuracy is 76.3%  
Feature set {8} was best. accuracy is 90.3% removing feature 7

Finished search !! The best feature subset is {7,8} which has an accuracy of 96.3%



Feature Count	Accuracy
1	0.75
2	0.827
3	0.896
4	0.813
5	0.817
6	0.847
7	0.963
8	0.903
9	0.820
10	0.723

Execution time 8083.5 seconds

Press 11 to run Forward Selection with small dataset  
Press 12 to run Forward Selection with large dataset  
Press 21 to run Backward Elimination with small dataset  
Press 22 to run Backward Elimination with large dataset  
Press any other key to exit  
exit  
Exit. Thank you