CS 205: Artificial Intelligence
Project 2

Name : Yash Aggarwal
SID : 862333037
Email : yagga004@ucr.edu
NetID : yagga004
Date : 14/June/2023

In completing this assignment I consulted:

1. The Project Statement Handout provided.
    a. https://d1u36hdvoy9y69.cloudfront.net/cs-205-ai/Project_2_feature_selection.pdf
2. For Real world dataset, Used the penguins dataset from https://github.com/allisonhorst/palmerpenguins

All the code is original **Except**

- Using the pandas module to handle datasets by creating dataframes
- Using numpy module for easy array manipulation and data frame handling
- Using urllib library to download the synthetic datasets
- Using zipfile module to extracts zip files
- Using the time module to track how long the functions run
- Using the OS module to save and store the datasets in OS
- Using the matplotlib module to plot the graphs
- Using the seaborn module to download the real world dataset
- Using sklearn to Convert categorical data to numerical data in real world dataset

Link to Code

- The code is available on github → https://github.com/yashUcr773/CS_205_AI/tree/main/Projects/Project%202
- The code can also be run on google colab → **https://colab.research.google.com/drive/1VyzhBqGJSqQLU3EI7kodLMSerO-ZY5Us**

Outline of this report

CS 205: Artificial Intelligence
Project 2: Feature Selection Using Nearest Neighbor Algorithm
Project Report

Name : Yash Aggarwal
SID: 862333037

1. Introduction

The Nearest Neighbor algorithm is a simple machine learning algorithm commonly used for classifying data points into different classes. The fundamental idea behind this algorithm is to measure the distance between an unknown datapoint and all the data points it has encountered during training. By finding the closest data point, the algorithm assigns the class label of that nearest point to the unknown data point.

The Nearest Neighbor algorithm is highly sensitive to outliers and irrelevant features present in the dataset. These outliers, which are data points significantly different from the majority, and irrelevant features that don't contribute meaningfully to the classification task, can introduce noise and negatively impact the algorithm's accuracy. To address this sensitivity and enhance the algorithm's performance, we need to select only the relevant features. By employing a feature search algorithm, we can identify and select only the most influential features while discarding the irrelevant ones. This process helps reduce the dimensionality of the dataset and focuses on the features that carry the most relevant information. Feature search algorithms, such as forward selection and backward elimination systematically evaluate different feature subsets to determine their impact on the algorithm's performance.

However, it is worth noting that the Nearest Neighbor algorithm itself does not incorporate a built-in feature selection mechanism. Feature selection is an independent step that can be applied as a preprocessing technique to enhance the algorithm's performance, especially when dealing with datasets containing numerous features or irrelevant attributes. Feature selection aims to identify the most informative and relevant features for a given task while discarding irrelevant or redundant ones. By reducing the dimensionality of the data, feature selection can improve the efficiency of the Nearest Neighbor algorithm and mitigate the impact of outlier and irrelevant features.

This report details the findings for the Implementation of Nearest Neighbor Classifier and using Forward Selection and Backward Elimination to find relevant features for Three types of datasets. As my data of birth is 17/July and I am implementing the project solo, I will be using the following datasets.

- CS170_small_Data__17.txt
    - 1000 instances with 10 features
- CS170_large_Data__17.txt
    - 2000 instances with 20 features
- CS170_XXXlarge_Data__14.txt
    - 4000 instances with 80 features

The report Also implements the Nearest Neighbor Algorithm on Real World Dataset. The dataset of choice is penguins dataset that classifies penguins into 3 species based on their physical attributes.

The project and report are a requirement for completion of the course CS 205: Artificial Intelligence taken under professor Dr. Eamonn Keogh in the Spring Quarter of 2023 at the University of California, Riverside.

Note: The accuracy graphs in this report have been scaled to be between 0 and 1 instead of 0 and 100 for easy plotting and visualization.

The Language of choice is Python (version 3.9.X) and additional imports include (pandas, numpy, urllib, zipfile, time, os, matplotlib, seaborn and sklearn). The Report also includes the original source code and link to run it.

2. Algorithms

The algorithms used for feature selection are

1) Forward Selection
2) Backward Elimination
3) Pruning In K-fold cross Validation

2.1 Forward Selection

Forward selection is a feature selection technique commonly used in machine learning. It is used to build a predictive model by iteratively adding the most relevant features to the model based on some evaluation criterion, accuracy measure using k-fold cross validation using nearest neighbor algorithm in our case.

The process of forward selection typically starts with an empty set of features and progressively adds one feature at a time. At each step, the algorithm evaluates the performance of the model using the current set of features, usually through a validation process such as cross-validation or holdout validation. The algorithm selects the feature that improves the model's performance the most and adds it to the feature set. This iterative process continues until a specified stopping criterion is met, such as reaching a predetermined number of features or when the addition of new features no longer improves the model's performance significantly.

The final selected features are then used to build the final predictive model. Forward selection has the advantage of being a simple and computationally efficient approach. It is particularly useful when dealing with a large number of potential features but limited computational resources or when trying to interpret the importance of individual features in the model.

2.2 Backward Elimination

Backward elimination is a feature selection technique used in machine learning. It is the reverse process of forward selection and aims to build a predictive model by iteratively removing the least relevant features from an initial full feature set. The process of backward elimination typically starts with a model that includes all available features. At each step, the algorithm evaluates the performance of the model using the current set of features, usually through a validation process such as cross-validation or holdout validation. The algorithm identifies the feature that contributes the least to the model's performance and removes it from the feature set. This iterative process continues until a specified stopping criterion is met, such as reaching a desired number of features or when removing additional features no longer significantly impacts the model's performance.

The final selected features are then used to build the final predictive model. Backward elimination can be advantageous when dealing with a large number of features, as it gradually reduces the feature set to a more manageable size. It can help eliminate irrelevant or redundant features, which can simplify the model and improve interpretability. Additionally, it can help mitigate the risk of overfitting that may occur when including too many features.

2.3 Pruning In K-fold cross Validation

K-Fold Cross validation is a valuable tool for assessing the accuracy and appropriateness of any algorithm. By examining the outcomes of k-fold cross validation, we can confidently evaluate whether the algorithm and feature selection are effective.

However, performing k-fold cross validation with multiple features and combinations can be time-consuming. To save time, we can halt the process if the desired model and feature combination fails to meet our requirements. To incorporate pruning into k-fold cross validation, we can establish a desired accuracy threshold that determines when a run should be terminated. We run the model for, let's say, 10 folds and calculate the average accuracy. If the accuracy falls below the threshold, we terminate the run; otherwise, we continue. This allows us to efficiently assess the performance of different models and feature combinations while minimizing the computational resources required.

3. Results

3.1 Small Dataset

The small dataset chosen is CS170_small_Data__17.txt and has 1000 instances with 10 features.
The Default Rate is: 80.80%

3.1.1 Forward Selection

When applying forward selection, we start with 80.80% accuracy for an empty set of features which is the default rate.
If we select feature 1, which is the best feature as per the search, the accuracy increases to 83.5%.
If we select feature 8 along with feature 1, which are the best features as per the search, the accuracy increases to 95.1%.

If we select feature 9 along with 1 and 8, which are the best features as per the search, the accuracy decreases slightly to 93.9%.
Going forward and completing the search, we can see that the max accuracy is 95.1% for features 1 and 8 and then the accuracy decreases as we go forward.

In figure 1, we can see the result of accuracy as a function of the number of features selected.

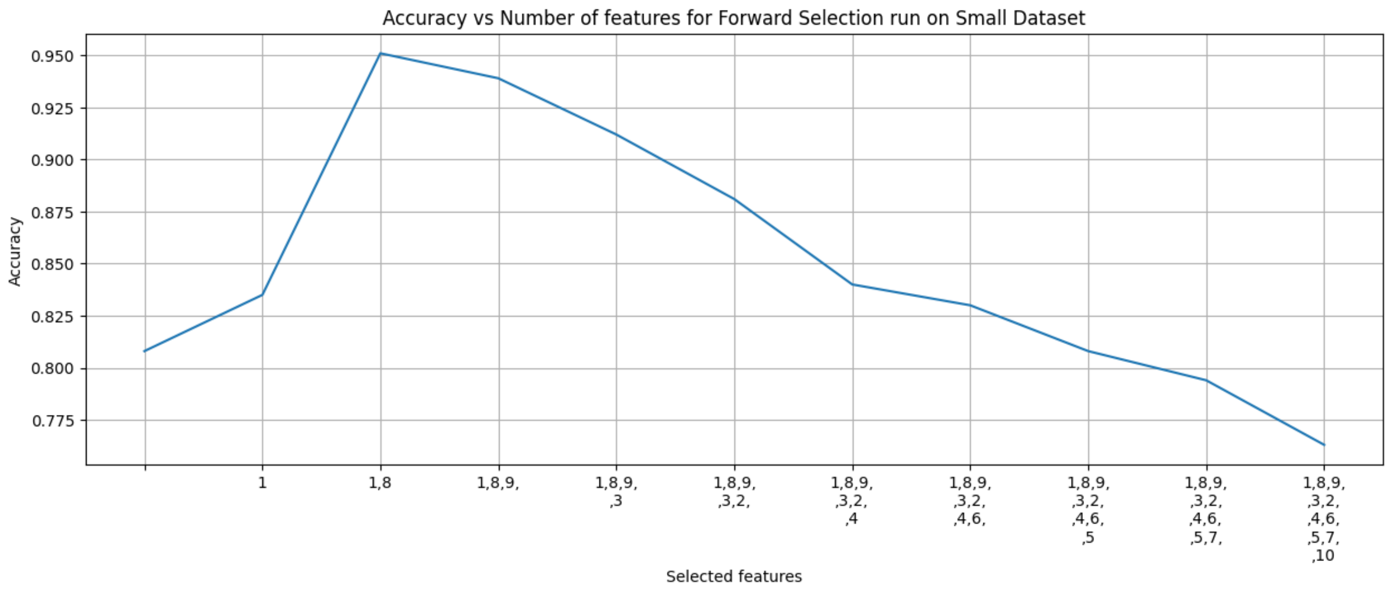


Figure 1. Accuracy for number of features selected in forward selection(scaled b/w 0-1)

3.1.2 Backward Elimination

When applying backward selection, we start with 76.3% accuracy for a superset of features or by selecting all the features.
If we remove feature 10, which is the worst feature as per the search, the accuracy increases to 79.4%.
If we remove feature 2 along with feature 10, which are the worst features as per the search, the accuracy decreases slightly to 78.4%.
If we remove feature 3 along with 2 and 10, which are the worst features as per the search, the accuracy increases to 81.9%
.
Going forward and completing the search, we can see that the max accuracy is 95.1% for features 1 and 8 and then the accuracy decreases as we go forward.

In figure 2, we can see the result of accuracy as a function of the number of features removed.

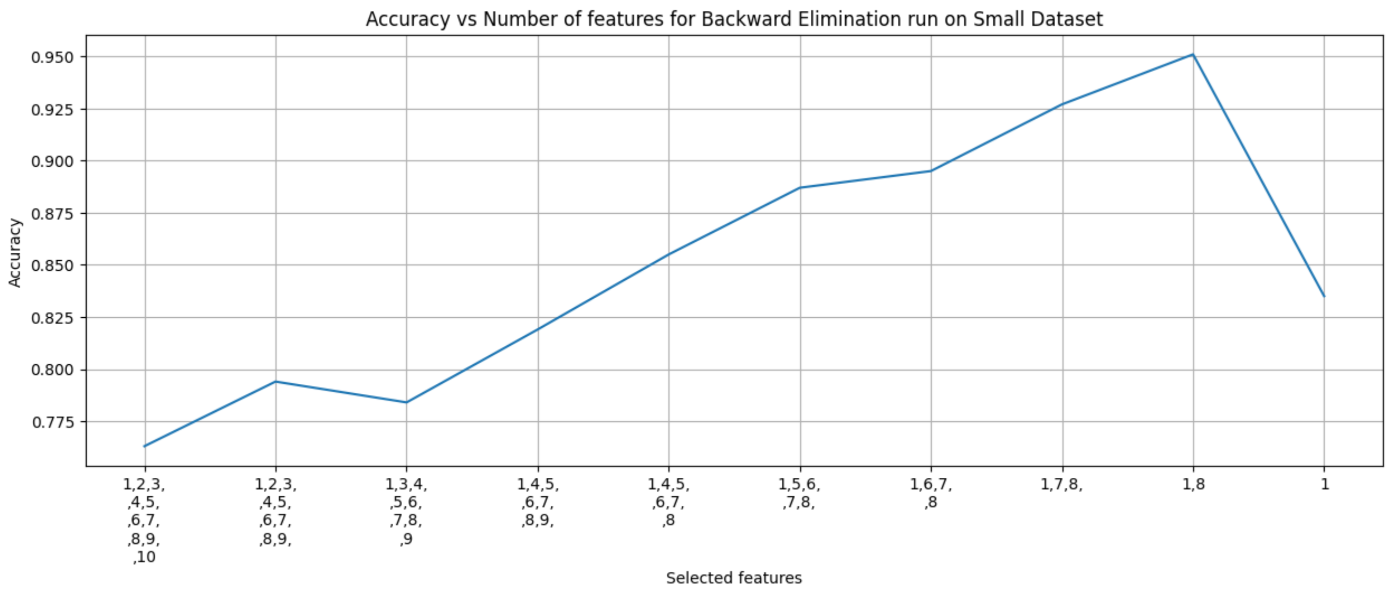


Figure 2. Accuracy for number of features selected in backward elimination(scaled b/w 0-1)

3.1.3 Computation Calculations

Using complete dataset without sampling and leave one out cross validation
Time to Run K-fold cross validation on 1 feature = 24 secs
Total number of combinations to run - n(n+1)/2 where n is 10 = 45
Total time to run for forward selection = 18 mins
Similarly, Total time to run for backward selection = 18 mins
However, we had implemented search pruning, total time it took to run the algorithms = 7 mins.

3.2 Large Dataset

The small dataset chosen is CS170_large_Data__17.txt and has 2000 instances with 20 features.
The Default Rate is: 82.30%

3.2.1 Forward Selection

When applying forward selection, we start with 82.30% accuracy for an empty set of features which is the default rate.
If we select feature 17, which is the best feature as per the search, the accuracy increases to 82.7%.
If we select feature 15 along with feature 17, which are the best features as per the search, the accuracy increases to 95.45%.
If we select feature 7 along with 17 and 15, which are the best features as per the search, the accuracy increases slightly to 95.95%.
Going forward and completing the search, we can see that the max accuracy is 95.95% for features 17, 7 and 15 and then the accuracy decreases as we go forward.

In figure 3, we can see the result of accuracy as a function of the number of features selected.

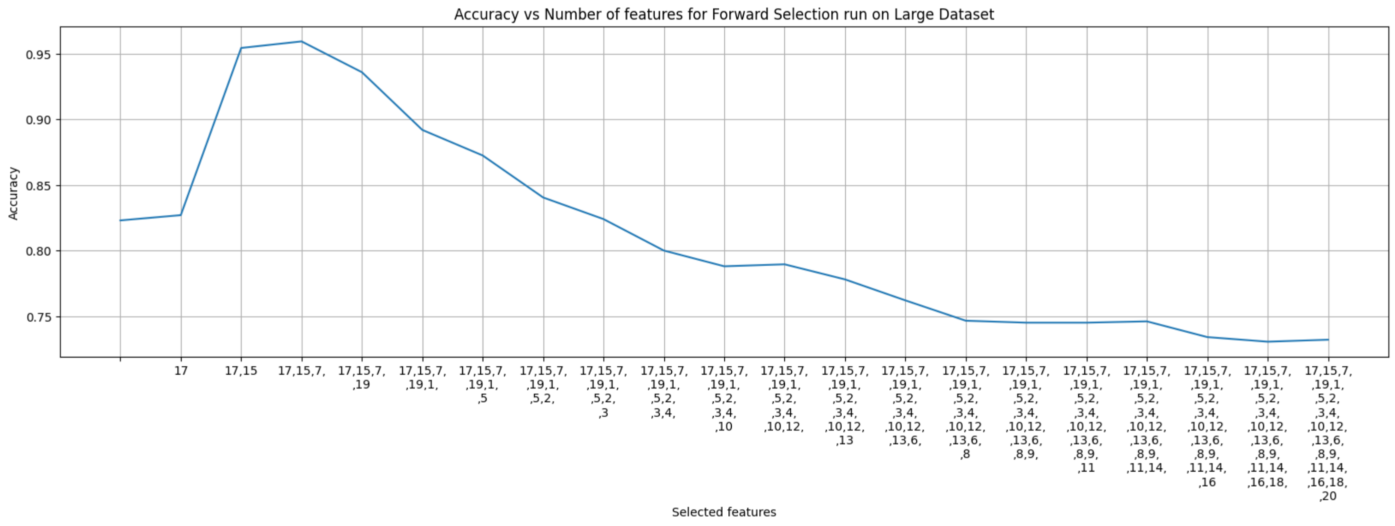


Figure 3. Accuracy for number of features selected in forward selection(scaled b/w 0-1)

3.2.2 Backward Elimination

When applying backward selection, we start with 73.2% accuracy for a superset of features or by selecting all the features.
If we remove feature 12, which is the worst feature as per the search, the accuracy increases to 73.7%.
If we remove feature 1 along with feature 12, which are the worst features as per the search, the accuracy decreases slightly to 73.4%.
If we remove feature 2 along with 12 and 1, which are the worst features as per the search, the accuracy increases to 73.95%
.
Going forward and completing the search, we can see that the max accuracy is 95.95% for features 15,17 and 7 and then the accuracy decreases as we go forward.

In figure 4, we can see the result of accuracy as a function of the number of features removed.

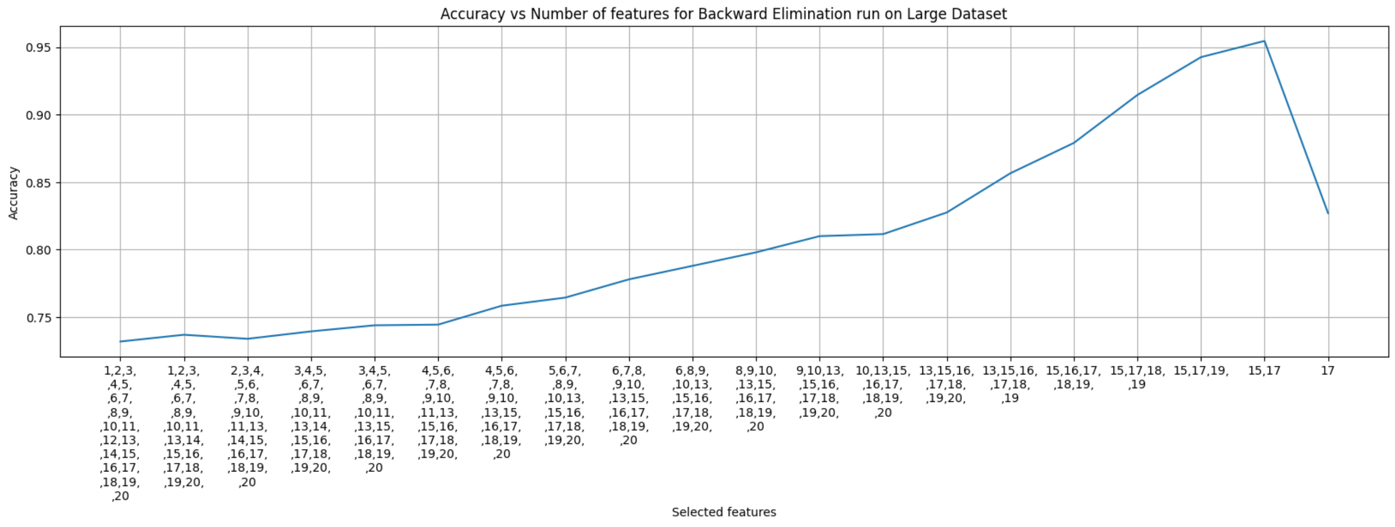


Figure 4. Accuracy for number of features selected in backward elimination(scaled b/w 0-1)

3.2.3 Computation Calculations

Using complete dataset without sampling and leave one out cross validation
Time to Run K-fold cross validation on 1 feature = 53 secs
Total number of combinations to run - n(n+1)/2 where n is 20 = 210
Total time to run for forward selection = 3 hrs and 8 mins

Similarly, Total time to run for backward selection = 3 hrs and 8 mins
However, we had implemented search pruning, total time it took to run the algorithms = 39 mins

3.3 XLarge Dataset

The XLarge dataset chosen is CS170_XXXlarge_Data__14.txt and has 4000 instances with 80 features.
The Default Rate is: 81.67%

3.3.1 Forward Selection

When applying forward selection, we start with 81.67% accuracy for an empty set of features which is the default rate.
If we select feature 60, which is the best feature as per the search, the accuracy increases to 84.28%.
If we select feature 35 along with feature 60, which are the best features as per the search, the accuracy increases to 97.47%.
If we select feature 41 along with 60 and 35, which are the best features as per the search, the accuracy decreases slightly to 96.83%.
Going forward and completing the search, we can see that the max accuracy is 97.47% for features 60 and 35 and then the accuracy decreases as we go forward.

In figure 5, we can see the result of accuracy as a function of the number of features selected.

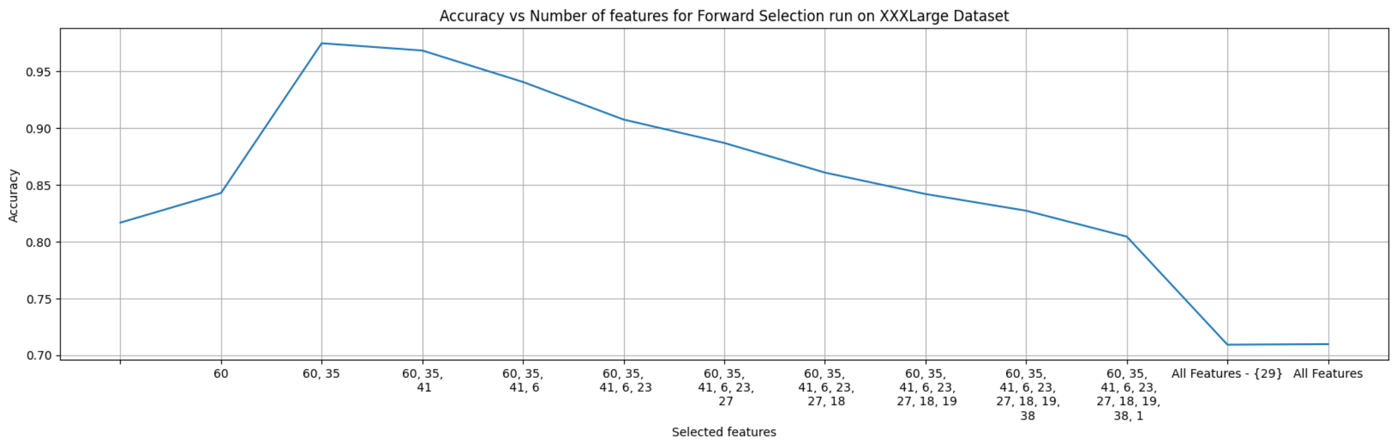


Figure 5. Accuracy for number of features selected in forward selection(scaled b/w 0-1)

3.3.2 Backward Elimination

When applying backward selection, we start with 67.25% accuracy for a superset of features or by selecting all the features.
If we remove feature 59, which is the worst feature as per the search, the accuracy increases to 70.02%.
If we remove feature 29 along with feature 59, which are the worst features as per the search, the accuracy increases slightly to 71.07%.
If we remove feature 38 along with 29 and 59, which are the worst features as per the search, the accuracy increases to 71.57%
.
Going forward and completing the search, we can see that the max accuracy is 97.47% for features 60 and 35 and then the accuracy decreases as we go forward.

In figure 6, we can see the result of accuracy as a function of the number of features removed.

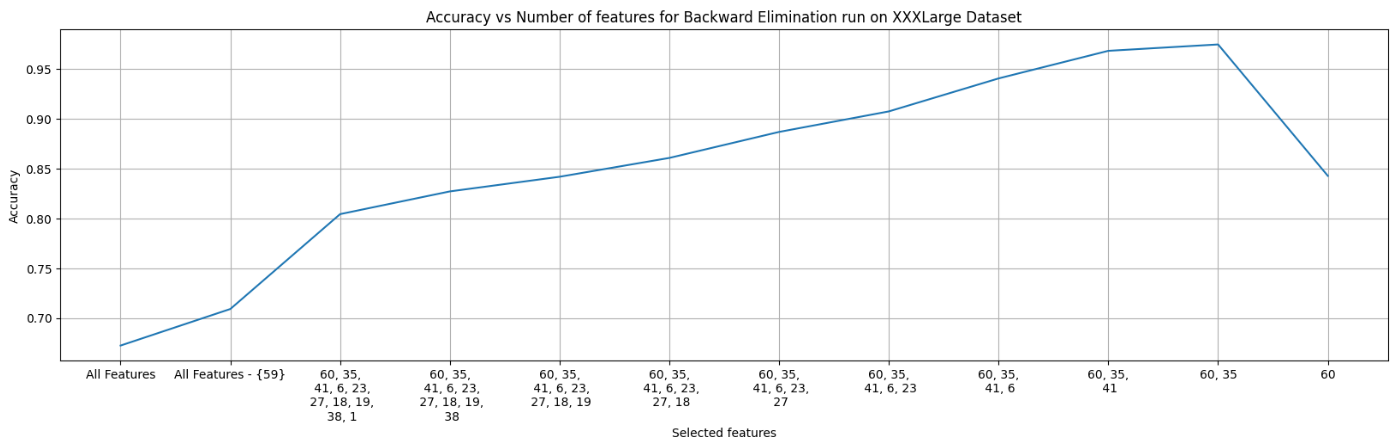


Figure 6. Accuracy for number of features selected in backward elimination(scaled b/w 0-1)

3.3.3 Computation Calculations

Using complete dataset without sampling and leave one out cross validation
Time to Run K-fold cross validation on 1 feature = 3 mins and 49 seconds
Total number of combinations to run - n(n+1)/2 where n is 80 = 3240
Total time to run for forward selection = 8 days 13 hours and 14 minutes
Similarly, Total time to run for backward selection = 8 days 13 hours and 14 minutes

Using complete dataset without sampling and use k=2 for k-fold cross validation
Time to Run K-fold cross validation on 1 feature = 1 min and 50 secs
Total number of combinations to run - n(n+1)/2 where n is 80 = 3240
Total time to run for forward selection = 4 days 2 hours and 7 minutes
Similarly, Total time to run for backward selection = 4 days 2 hours and 7 minutes

Sample the dataset and use only 1200 out of 4000 and use leave one out cross validation
Time to Run K-fold cross validation on 1 feature = 20 secs
Total number of combinations to run - n(n+1)/2 where n is 80 = 3240
Total time to run for forward selection = 18 hrs and 28 minutes
Similarly, Total time to run for backward selection = 18 hrs and 28 minutes

Sample the dataset and use only 1200 out of 4000 and use use k=2 for k-fold cross validation
Time to Run K-fold cross validation on 1 feature = 17 secs
Total number of combinations to run - n(n+1)/2 where n is 80 = 3240
Total time to run for forward selection = 15 hrs and 8 minutes
Similarly, Total time to run for backward selection = 15 hrs and 8 minutes

However, we had implemented search pruning, total time it took to run the algorithms = 10 hours and 51 mins

4. Implementation on Real World Dataset

Used the Penguins Dataset from https://github.com/allisonhorst/palmerpenguins that contains species of penguins divided in three classes. There are 6 features that can be used to predict the class.

The dataset contains the following fields.
Target
- **Species**: penguin species (Chinstrap, Adélie, or Gentoo)

Features

1. **Island**: island name (Dream, Torgersen, or Biscoe) in the Palmer Archipelago (Antarctica)
2. **Culmen_length_mm**: culmen length (mm)
3. **Culmen_depth_mm**: culmen depth (mm)
4. **Flipper_length_mm**: flipper length (mm)
5. **Body_mass_g**: body mass (g)
6. **Sex**: penguin sex

The dataset has 333 instances with 6 features.
The Default Rate is: 43.84%

4.1 Forward Selection

When applying forward selection, we start with 43.84% accuracy for an empty set of features which is the default rate.
If we select feature 4, which is the best feature as per the search, the accuracy increases to 76.28%.
If we select feature 2 along with feature 4, which are the best features as per the search, the accuracy increases to 93.69%.
If we select feature 3 along with 2 and 4, which are the best features as per the search, the accuracy increases slightly to 96.40%.
Going forward and completing the search, we can see that the max accuracy is 96.70% for features 4,2,3 and 1 and then the accuracy decreases as we go forward.

Therefore we can say that Flipper_length_mm, Culmen_length_mm, Culmen_depth_mm, Island are the best features to estimate the species of the penguin.

In figure 7, we can see the result of accuracy as a function of the number of features selected.

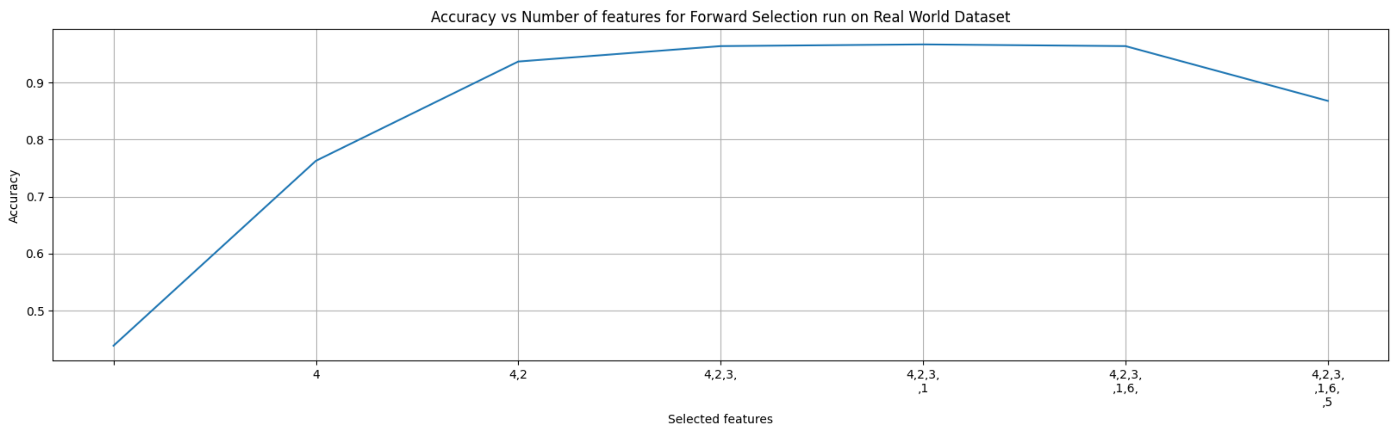


Figure 7. Accuracy for number of features selected in forward selection(scaled b/w 0-1)

4.2 Backward Elimination

When applying backward selection, we start with 86.79% accuracy for a superset of features or by selecting all the features.
If we remove feature 5, which is the worst feature as per the search, the accuracy increases to 96.40%.
If we remove feature 4 along with feature 5, which are the worst features as per the search, the accuracy increases to 97.90%.
If we remove feature 1 along with 4 and 5, which are the worst features as per the search, the accuracy decreases to 97.60%
.

Going forward and completing the search, we can see that the max accuracy is 96.70% for features 4,2,3 and 1 and then the accuracy decreases as we go forward.

Therefore we can say that Flipper_length_mm, Culmen_length_mm, Culmen_depth_mm, Island are the best features to estimate the species of the penguin.

In figure 8, we can see the result of accuracy as a function of the number of features removed.

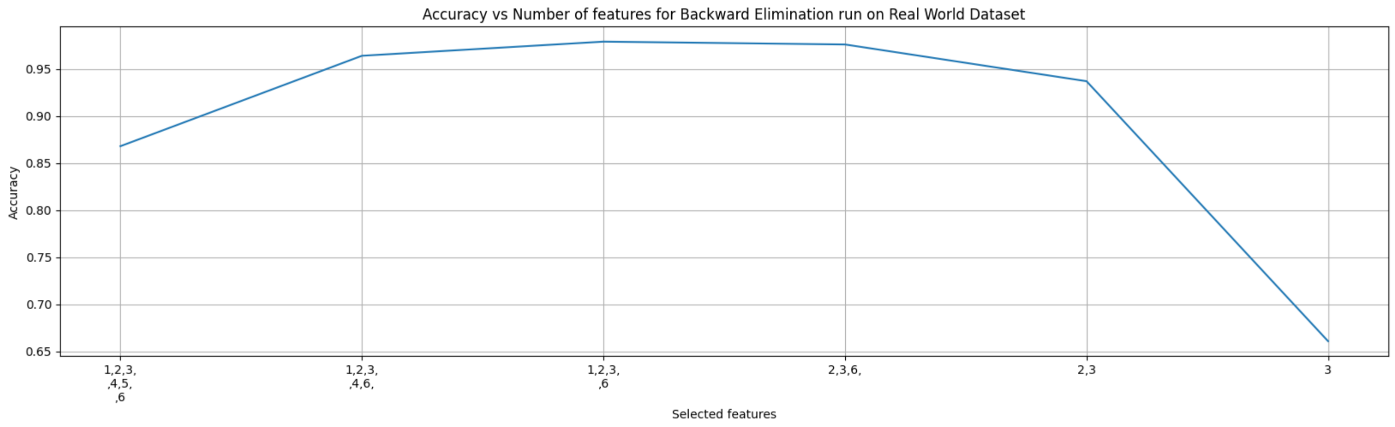


Figure 8. Accuracy for number of features selected in backward elimination(scaled b/w 0-1)

4.3 Computation Calculations

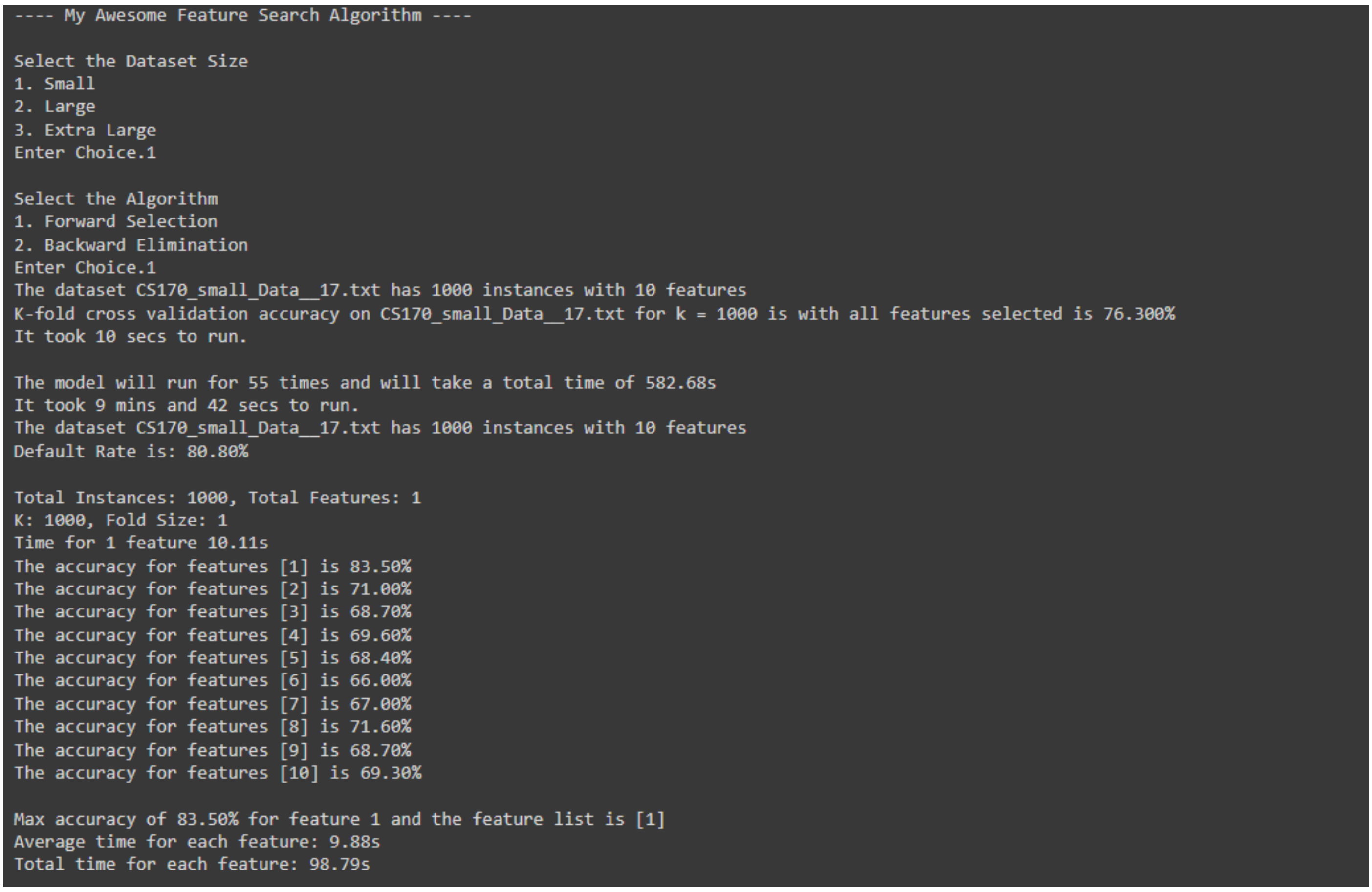
Using complete dataset without sampling and leave one out cross validation
Time to Run K-fold cross validation on 1 feature = 2 secs
Total number of combinations to run - n(n+1)/2 where n is 6 = 21
Total time to run for forward selection = 48 secs
Similarly, Total time to run for backward selection = 48 secs
However, we had implemented search pruning, total time it took to run the algorithms = 33 secs

5. Conclusions

1. For the Small Dataset, The best features to choose are 1 and 8 and the best accuracy is 95.1%. This can be seen using both the forward selection and backward elimination.
2. For the Large Dataset, The best features to choose are 17, 7 and 15 and and the best accuracy is 95.1%. This can be seen using both the forward selection and backward elimination.
3. For the XXXLarge Dataset, The best features to choose are 60 and 35 and and the best accuracy is 97.47%. This can be seen using both the forward selection and backward elimination.
4. For the Real World Dataset, The best features to choose are Flipper_length_mm, Culmen_length_mm, Culmen_depth_mm, Island and the best accuracy is 96.70%. This can be seen using both the forward selection and backward elimination.

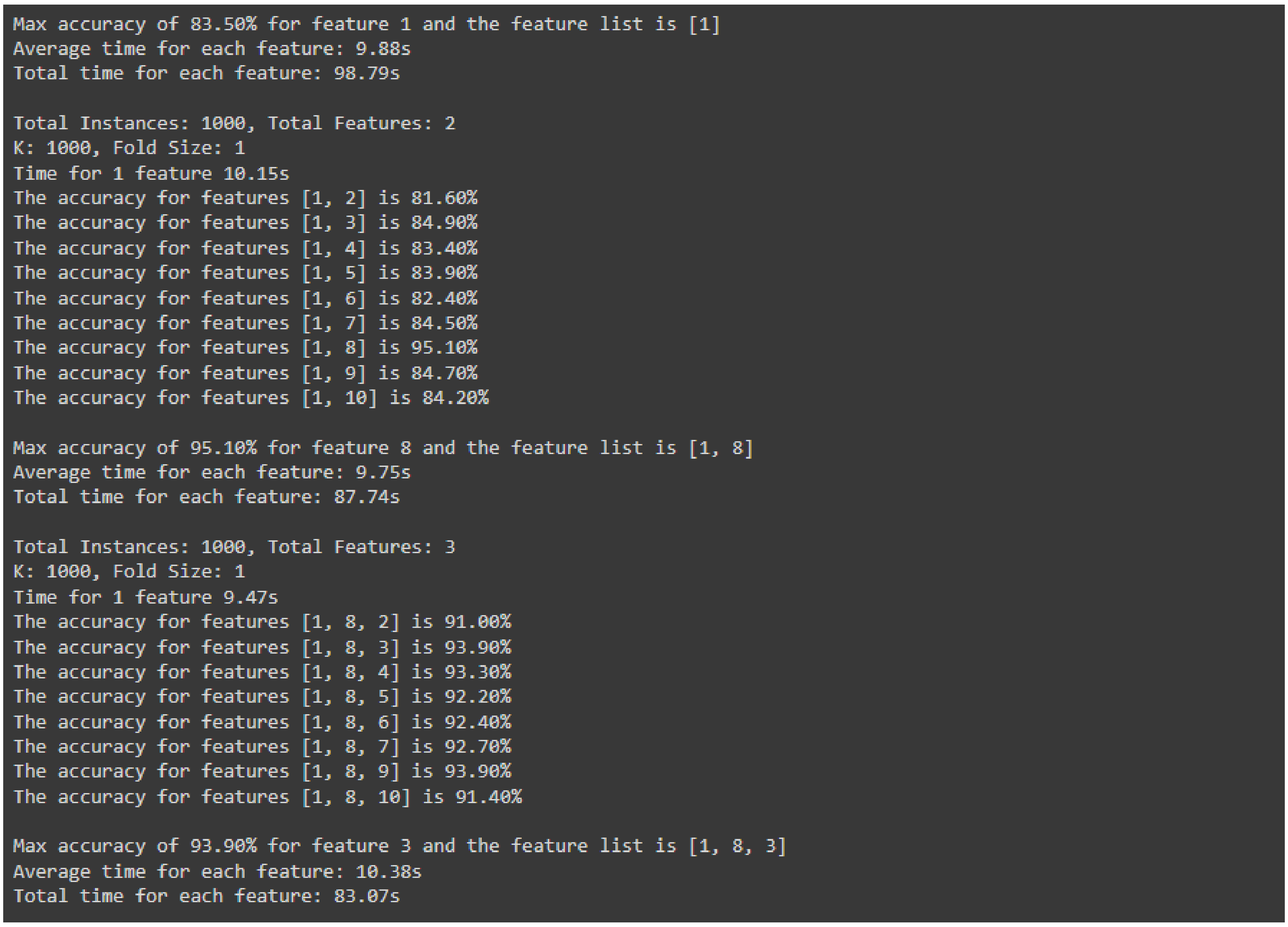
6. Traceback of Forward Selection

Figures 9, 10, 11, 12 and 13 show the traceback on an easy dataset using the forward selection algorithm.



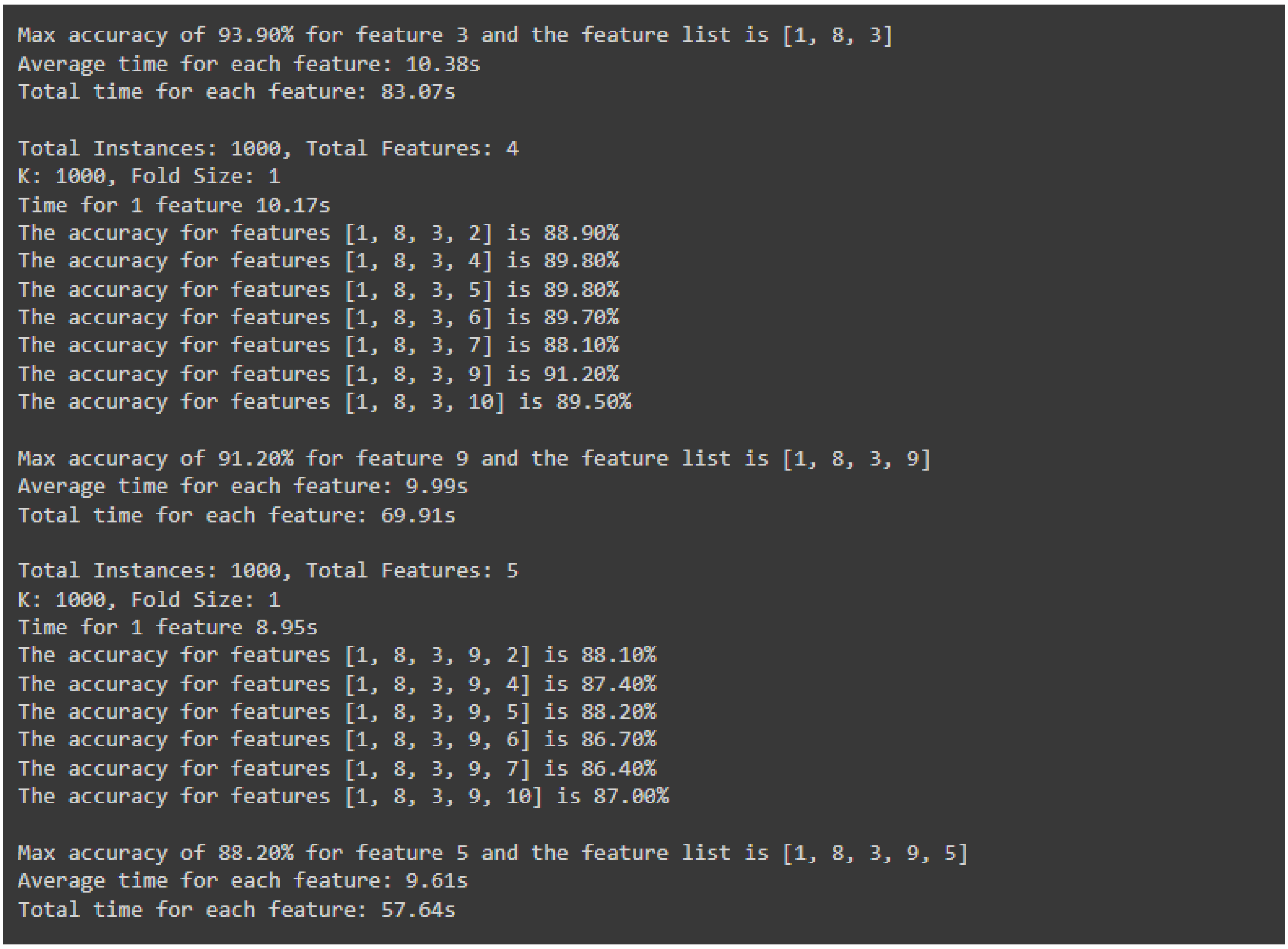
```
---- My Awesome Feature Search Algorithm ----

Select the Dataset Size
1. Small
2. Large
3. Extra Large
Enter Choice.1

Select the Algorithm
1. Forward Selection
2. Backward Elimination
Enter Choice.1
The dataset CS170_small_Data__17.txt has 1000 instances with 10 features
K-fold cross validation accuracy on CS170_small_Data__17.txt for k = 1000 is with all features selected is 76.300%
It took 10 secs to run.

The model will run for 55 times and will take a total time of 582.68s
It took 9 mins and 42 secs to run.
The dataset CS170_small_Data__17.txt has 1000 instances with 10 features
Default Rate is: 80.80%

Total Instances: 1000, Total Features: 1
K: 1000, Fold Size: 1
Time for 1 feature 10.11s
The accuracy for features [1] is 83.50%
The accuracy for features [2] is 71.00%
The accuracy for features [3] is 68.70%
The accuracy for features [4] is 69.60%
The accuracy for features [5] is 68.40%
The accuracy for features [6] is 66.00%
The accuracy for features [7] is 67.00%
The accuracy for features [8] is 71.60%
The accuracy for features [9] is 68.70%
The accuracy for features [10] is 69.30%

Max accuracy of 83.50% for feature 1 and the feature list is [1]
Average time for each feature: 9.88s
Total time for each feature: 98.79s
```

Figure 9: Start of Traceback on easy puzzle with forward selection algorithm. Here, the first best feature is feature 1.



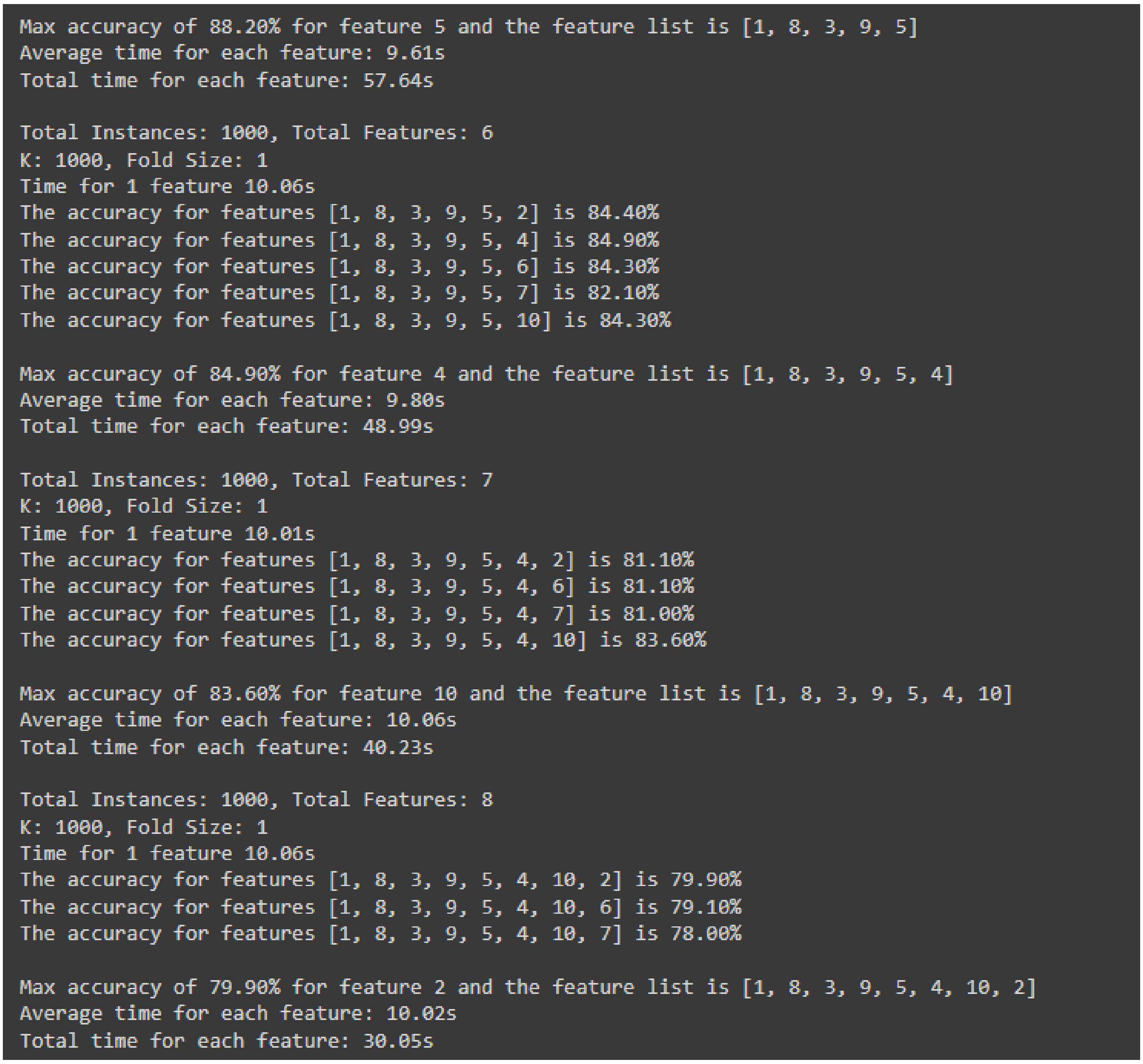
```
Max accuracy of 83.50% for feature 1 and the feature list is [1]
Average time for each feature: 9.88s
Total time for each feature: 98.79s

Total Instances: 1000, Total Features: 2
K: 1000, Fold Size: 1
Time for 1 feature 10.15s
The accuracy for features [1, 2] is 81.60%
The accuracy for features [1, 3] is 84.90%
The accuracy for features [1, 4] is 83.40%
The accuracy for features [1, 5] is 83.90%
The accuracy for features [1, 6] is 82.40%
The accuracy for features [1, 7] is 84.50%
The accuracy for features [1, 8] is 95.10%
The accuracy for features [1, 9] is 84.70%
The accuracy for features [1, 10] is 84.20%

Max accuracy of 95.10% for feature 8 and the feature list is [1, 8]
Average time for each feature: 9.75s
Total time for each feature: 87.74s

Total Instances: 1000, Total Features: 3
K: 1000, Fold Size: 1
Time for 1 feature 9.47s
The accuracy for features [1, 8, 2] is 91.00%
The accuracy for features [1, 8, 3] is 93.90%
The accuracy for features [1, 8, 4] is 93.30%
The accuracy for features [1, 8, 5] is 92.20%
The accuracy for features [1, 8, 6] is 92.40%
The accuracy for features [1, 8, 7] is 92.70%
The accuracy for features [1, 8, 9] is 93.90%
The accuracy for features [1, 8, 10] is 91.40%

Max accuracy of 93.90% for feature 3 and the feature list is [1, 8, 3]
Average time for each feature: 10.38s
Total time for each feature: 83.07s
```

Figure 10: Continuation of the Traceback. Here, the second best feature is feature 8 and The third best feature is feature 3.



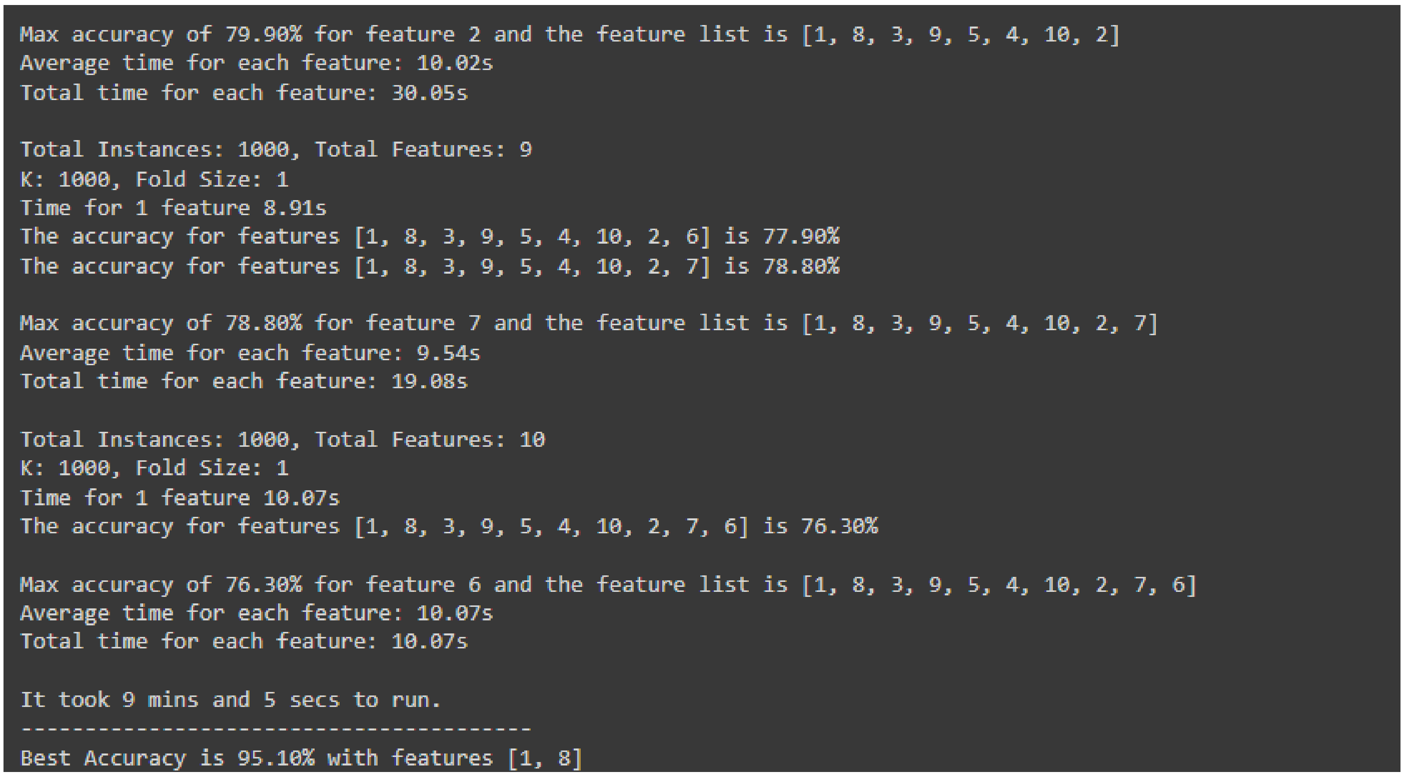
```
Max accuracy of 93.90% for feature 3 and the feature list is [1, 8, 3]
Average time for each feature: 10.38s
Total time for each feature: 83.07s

Total Instances: 1000, Total Features: 4
K: 1000, Fold Size: 1
Time for 1 feature 10.17s
The accuracy for features [1, 8, 3, 2] is 88.90%
The accuracy for features [1, 8, 3, 4] is 89.80%
The accuracy for features [1, 8, 3, 5] is 89.80%
The accuracy for features [1, 8, 3, 6] is 89.70%
The accuracy for features [1, 8, 3, 7] is 88.10%
The accuracy for features [1, 8, 3, 9] is 91.20%
The accuracy for features [1, 8, 3, 10] is 89.50%

Max accuracy of 91.20% for feature 9 and the feature list is [1, 8, 3, 9]
Average time for each feature: 9.99s
Total time for each feature: 69.91s

Total Instances: 1000, Total Features: 5
K: 1000, Fold Size: 1
Time for 1 feature 8.95s
The accuracy for features [1, 8, 3, 9, 2] is 88.10%
The accuracy for features [1, 8, 3, 9, 4] is 87.40%
The accuracy for features [1, 8, 3, 9, 5] is 88.20%
The accuracy for features [1, 8, 3, 9, 6] is 86.70%
The accuracy for features [1, 8, 3, 9, 7] is 86.40%
The accuracy for features [1, 8, 3, 9, 10] is 87.00%

Max accuracy of 88.20% for feature 5 and the feature list is [1, 8, 3, 9, 5]
Average time for each feature: 9.61s
Total time for each feature: 57.64s
```

Figure 11: Continuation of the Traceback. Here, the fourth best feature is feature 9 and The fifth best feature is feature 5.



```
Max accuracy of 88.20% for feature 5 and the feature list is [1, 8, 3, 9, 5]
Average time for each feature: 9.61s
Total time for each feature: 57.64s

Total Instances: 1000, Total Features: 6
K: 1000, Fold Size: 1
Time for 1 feature 10.06s
The accuracy for features [1, 8, 3, 9, 5, 2] is 84.40%
The accuracy for features [1, 8, 3, 9, 5, 4] is 84.90%
The accuracy for features [1, 8, 3, 9, 5, 6] is 84.30%
The accuracy for features [1, 8, 3, 9, 5, 7] is 82.10%
The accuracy for features [1, 8, 3, 9, 5, 10] is 84.30%

Max accuracy of 84.90% for feature 4 and the feature list is [1, 8, 3, 9, 5, 4]
Average time for each feature: 9.80s
Total time for each feature: 48.99s

Total Instances: 1000, Total Features: 7
K: 1000, Fold Size: 1
Time for 1 feature 10.01s
The accuracy for features [1, 8, 3, 9, 5, 4, 2] is 81.10%
The accuracy for features [1, 8, 3, 9, 5, 4, 6] is 81.10%
The accuracy for features [1, 8, 3, 9, 5, 4, 7] is 81.00%
The accuracy for features [1, 8, 3, 9, 5, 4, 10] is 83.60%

Max accuracy of 83.60% for feature 10 and the feature list is [1, 8, 3, 9, 5, 4, 10]
Average time for each feature: 10.06s
Total time for each feature: 40.23s

Total Instances: 1000, Total Features: 8
K: 1000, Fold Size: 1
Time for 1 feature 10.06s
The accuracy for features [1, 8, 3, 9, 5, 4, 10, 2] is 79.90%
The accuracy for features [1, 8, 3, 9, 5, 4, 10, 6] is 79.10%
The accuracy for features [1, 8, 3, 9, 5, 4, 10, 7] is 78.00%

Max accuracy of 79.90% for feature 2 and the feature list is [1, 8, 3, 9, 5, 4, 10, 2]
Average time for each feature: 10.02s
Total time for each feature: 30.05s
```

Figure 12: Continuation of the Traceback. Here, the sixth best feature is feature 4 and The seventh best feature is feature 10 and the eight best feature is feature 2.

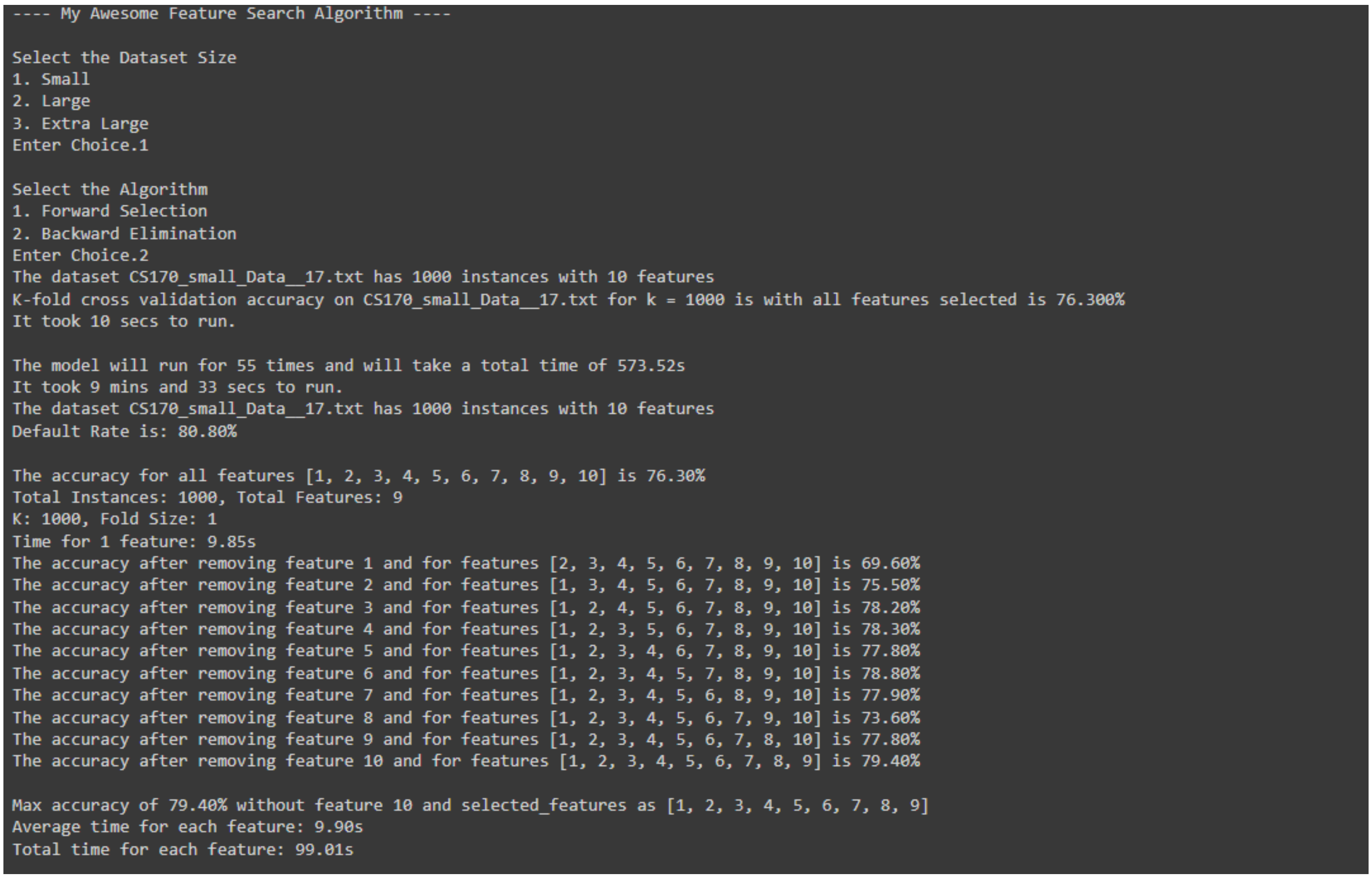


```
Max accuracy of 79.90% for feature 2 and the feature list is [1, 8, 3, 9, 5, 4, 10, 2]
Average time for each feature: 10.02s
Total time for each feature: 30.05s

Total Instances: 1000, Total Features: 9
K: 1000, Fold Size: 1
Time for 1 feature 8.91s
The accuracy for features [1, 8, 3, 9, 5, 4, 10, 2, 6] is 77.90%
The accuracy for features [1, 8, 3, 9, 5, 4, 10, 2, 7] is 78.80%

Max accuracy of 78.80% for feature 7 and the feature list is [1, 8, 3, 9, 5, 4, 10, 2, 7]
Average time for each feature: 9.54s
Total time for each feature: 19.08s

Total Instances: 1000, Total Features: 10
K: 1000, Fold Size: 1
Time for 1 feature 10.07s
The accuracy for features [1, 8, 3, 9, 5, 4, 10, 2, 7, 6] is 76.30%

Max accuracy of 76.30% for feature 6 and the feature list is [1, 8, 3, 9, 5, 4, 10, 2, 7, 6]
Average time for each feature: 10.07s
Total time for each feature: 10.07s

It took 9 mins and 5 secs to run.
----------------------------------------
Best Accuracy is 95.10% with features [1, 8]
```

Figure 13: Continuation of the Traceback. Here, the ninth best feature is feature 7 and final feature is 6.

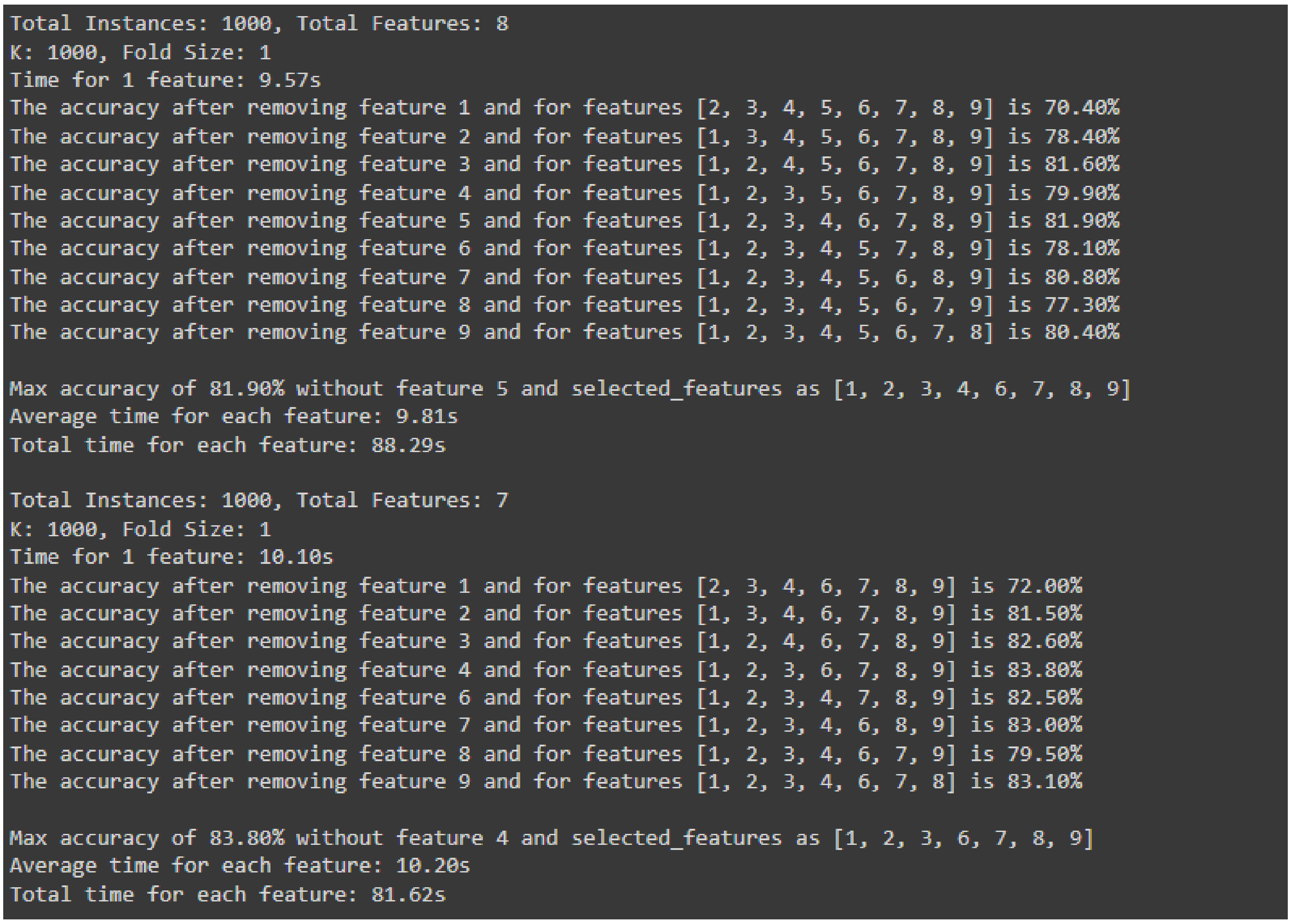
7. Traceback of Backward Elimination

Figures 14, 15, 16, 17 and 18 show the traceback on an easy dataset using the forward selection algorithm .



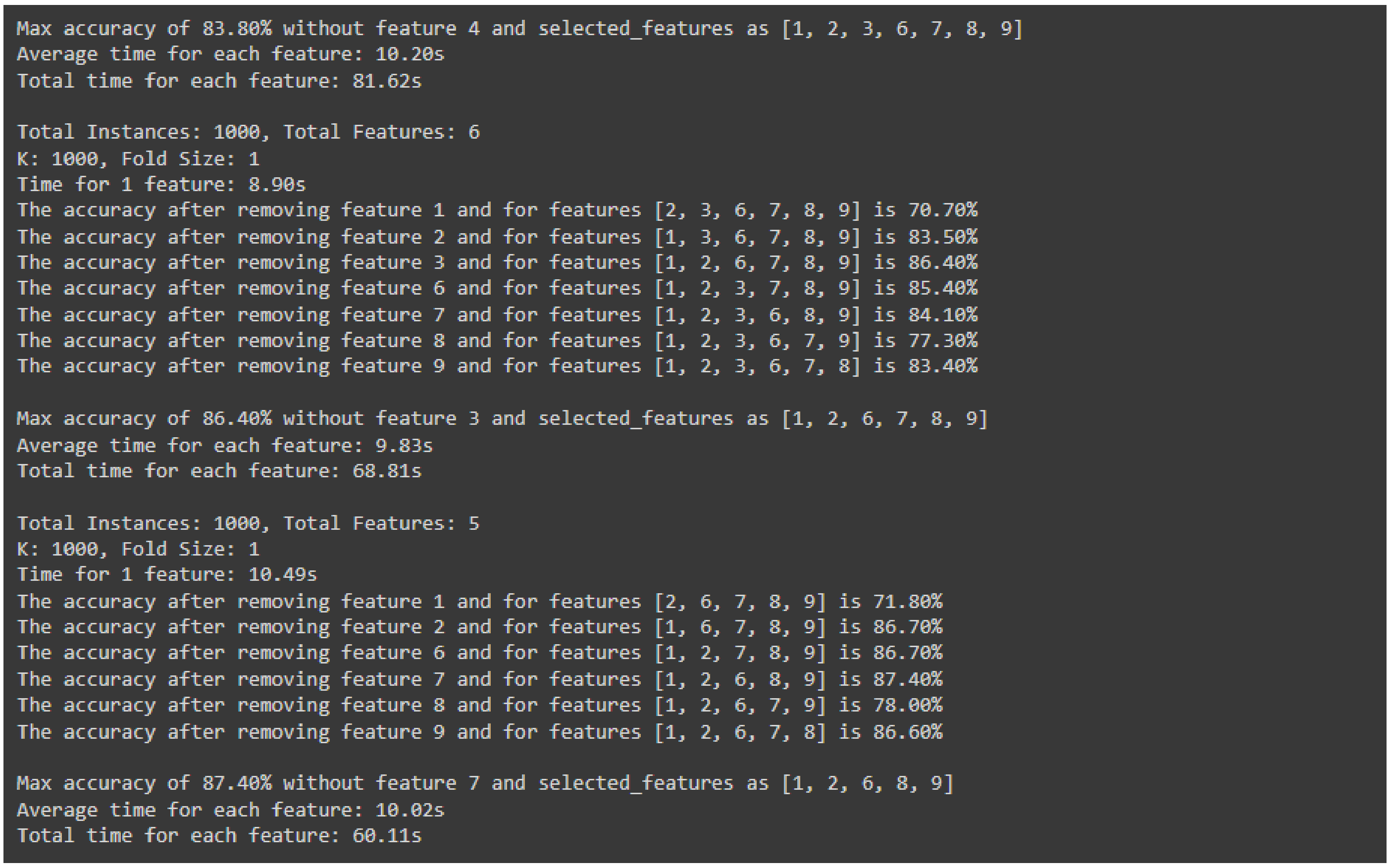
```
---- My Awesome Feature Search Algorithm ----

Select the Dataset Size
1. Small
2. Large
3. Extra Large
Enter Choice.1

Select the Algorithm
1. Forward Selection
2. Backward Elimination
Enter Choice.2
The dataset CS170_small_Data__17.txt has 1000 instances with 10 features
K-fold cross validation accuracy on CS170_small_Data__17.txt for k = 1000 is with all features selected is 76.300%
It took 10 secs to run.

The model will run for 55 times and will take a total time of 573.52s
It took 9 mins and 33 secs to run.
The dataset CS170_small_Data__17.txt has 1000 instances with 10 features
Default Rate is: 80.80%

The accuracy for all features [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] is 76.30%
Total Instances: 1000, Total Features: 9
K: 1000, Fold Size: 1
Time for 1 feature: 9.85s
The accuracy after removing feature 1 and for features [2, 3, 4, 5, 6, 7, 8, 9, 10] is 69.60%
The accuracy after removing feature 2 and for features [1, 3, 4, 5, 6, 7, 8, 9, 10] is 75.50%
The accuracy after removing feature 3 and for features [1, 2, 4, 5, 6, 7, 8, 9, 10] is 78.20%
The accuracy after removing feature 4 and for features [1, 2, 3, 5, 6, 7, 8, 9, 10] is 78.30%
The accuracy after removing feature 5 and for features [1, 2, 3, 4, 6, 7, 8, 9, 10] is 77.80%
The accuracy after removing feature 6 and for features [1, 2, 3, 4, 5, 7, 8, 9, 10] is 78.80%
The accuracy after removing feature 7 and for features [1, 2, 3, 4, 5, 6, 8, 9, 10] is 77.90%
The accuracy after removing feature 8 and for features [1, 2, 3, 4, 5, 6, 7, 9, 10] is 73.60%
The accuracy after removing feature 9 and for features [1, 2, 3, 4, 5, 6, 7, 8, 10] is 77.80%
The accuracy after removing feature 10 and for features [1, 2, 3, 4, 5, 6, 7, 8, 9] is 79.40%

Max accuracy of 79.40% without feature 10 and selected_features as [1, 2, 3, 4, 5, 6, 7, 8, 9]
Average time for each feature: 9.90s
Total time for each feature: 99.01s
```

Figure 14: Start of Traceback on easy puzzle with backward elimination algorithm. Here, the first worst feature to remove is feature 10.



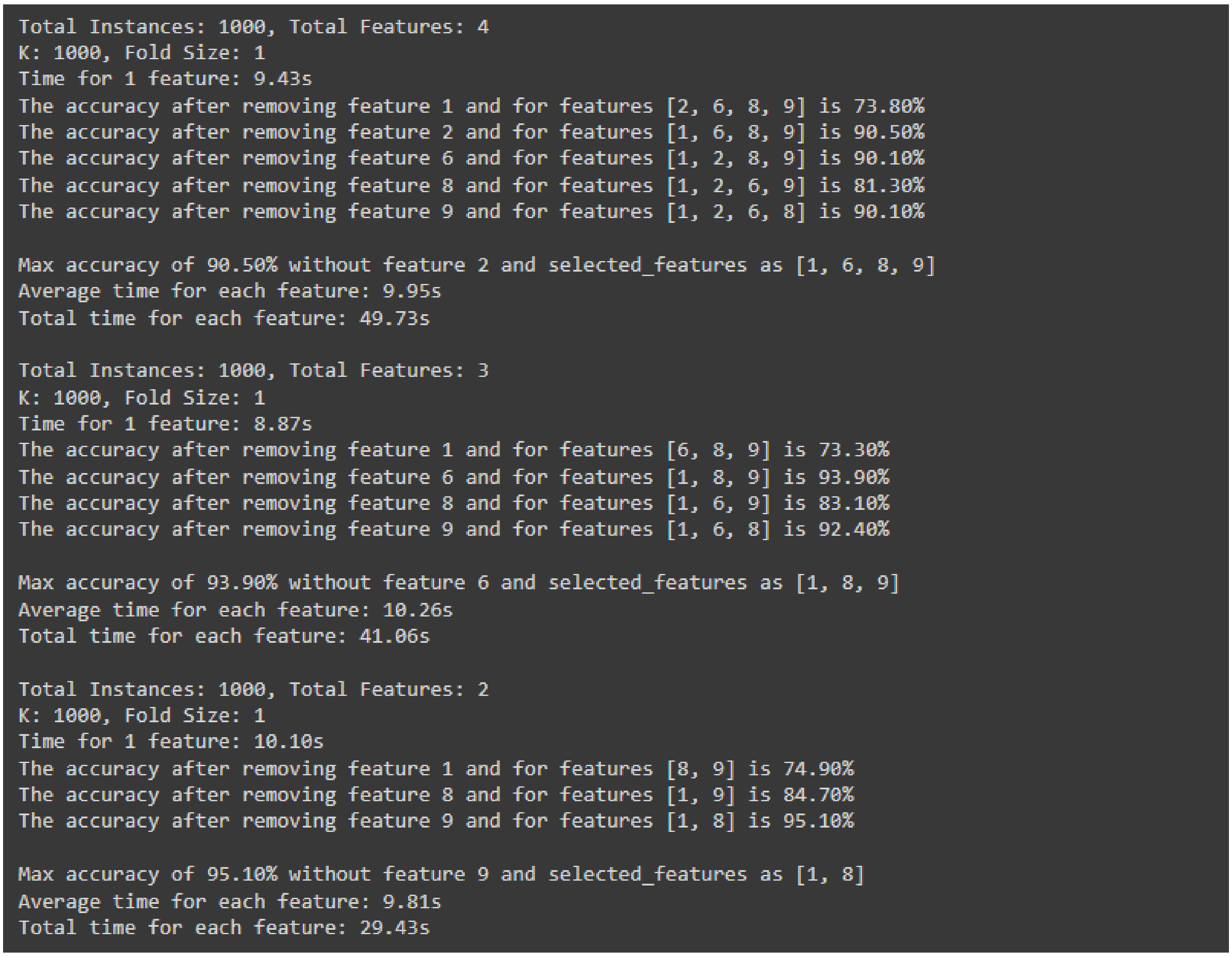
```
Total Instances: 1000, Total Features: 8
K: 1000, Fold Size: 1
Time for 1 feature: 9.57s
The accuracy after removing feature 1 and for features [2, 3, 4, 5, 6, 7, 8, 9] is 70.40%
The accuracy after removing feature 2 and for features [1, 3, 4, 5, 6, 7, 8, 9] is 78.40%
The accuracy after removing feature 3 and for features [1, 2, 4, 5, 6, 7, 8, 9] is 81.60%
The accuracy after removing feature 4 and for features [1, 2, 3, 5, 6, 7, 8, 9] is 79.90%
The accuracy after removing feature 5 and for features [1, 2, 3, 4, 6, 7, 8, 9] is 81.90%
The accuracy after removing feature 6 and for features [1, 2, 3, 4, 5, 7, 8, 9] is 78.10%
The accuracy after removing feature 7 and for features [1, 2, 3, 4, 5, 6, 8, 9] is 80.80%
The accuracy after removing feature 8 and for features [1, 2, 3, 4, 5, 6, 7, 9] is 77.30%
The accuracy after removing feature 9 and for features [1, 2, 3, 4, 5, 6, 7, 8] is 80.40%

Max accuracy of 81.90% without feature 5 and selected_features as [1, 2, 3, 4, 6, 7, 8, 9]
Average time for each feature: 9.81s
Total time for each feature: 88.29s

Total Instances: 1000, Total Features: 7
K: 1000, Fold Size: 1
Time for 1 feature: 10.10s
The accuracy after removing feature 1 and for features [2, 3, 4, 6, 7, 8, 9] is 72.00%
The accuracy after removing feature 2 and for features [1, 3, 4, 6, 7, 8, 9] is 81.50%
The accuracy after removing feature 3 and for features [1, 2, 4, 6, 7, 8, 9] is 82.60%
The accuracy after removing feature 4 and for features [1, 2, 3, 6, 7, 8, 9] is 83.80%
The accuracy after removing feature 6 and for features [1, 2, 3, 4, 7, 8, 9] is 82.50%
The accuracy after removing feature 7 and for features [1, 2, 3, 4, 6, 8, 9] is 83.00%
The accuracy after removing feature 8 and for features [1, 2, 3, 4, 6, 7, 9] is 79.50%
The accuracy after removing feature 9 and for features [1, 2, 3, 4, 6, 7, 8] is 83.10%

Max accuracy of 83.80% without feature 4 and selected_features as [1, 2, 3, 6, 7, 8, 9]
Average time for each feature: 10.20s
Total time for each feature: 81.62s
```

Figure 15: Continuation of the Traceback. Here, the second worst feature to remove is feature 5. The third worst feature to remove is Feature 4



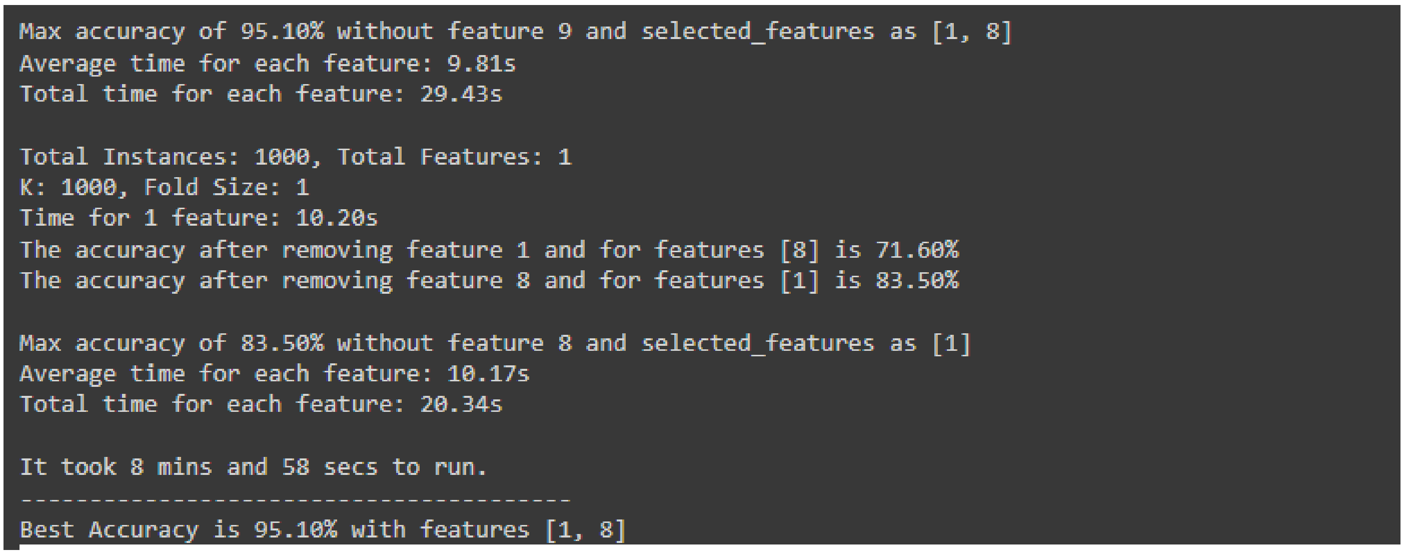
```
Max accuracy of 83.80% without feature 4 and selected_features as [1, 2, 3, 6, 7, 8, 9]
Average time for each feature: 10.20s
Total time for each feature: 81.62s

Total Instances: 1000, Total Features: 6
K: 1000, Fold Size: 1
Time for 1 feature: 8.90s
The accuracy after removing feature 1 and for features [2, 3, 6, 7, 8, 9] is 70.70%
The accuracy after removing feature 2 and for features [1, 3, 6, 7, 8, 9] is 83.50%
The accuracy after removing feature 3 and for features [1, 2, 6, 7, 8, 9] is 86.40%
The accuracy after removing feature 6 and for features [1, 2, 3, 7, 8, 9] is 85.40%
The accuracy after removing feature 7 and for features [1, 2, 3, 6, 8, 9] is 84.10%
The accuracy after removing feature 8 and for features [1, 2, 3, 6, 7, 9] is 77.30%
The accuracy after removing feature 9 and for features [1, 2, 3, 6, 7, 8] is 83.40%

Max accuracy of 86.40% without feature 3 and selected_features as [1, 2, 6, 7, 8, 9]
Average time for each feature: 9.83s
Total time for each feature: 68.81s

Total Instances: 1000, Total Features: 5
K: 1000, Fold Size: 1
Time for 1 feature: 10.49s
The accuracy after removing feature 1 and for features [2, 6, 7, 8, 9] is 71.80%
The accuracy after removing feature 2 and for features [1, 6, 7, 8, 9] is 86.70%
The accuracy after removing feature 6 and for features [1, 2, 7, 8, 9] is 86.70%
The accuracy after removing feature 7 and for features [1, 2, 6, 8, 9] is 87.40%
The accuracy after removing feature 8 and for features [1, 2, 6, 7, 9] is 78.00%
The accuracy after removing feature 9 and for features [1, 2, 6, 7, 8] is 86.60%

Max accuracy of 87.40% without feature 7 and selected_features as [1, 2, 6, 8, 9]
Average time for each feature: 10.02s
Total time for each feature: 60.11s
```

Figure 16: Continuation of the Traceback. Here, the fourth worst feature to remove is feature 3. The fifth worst feature to remove is Feature 7



```
Total Instances: 1000, Total Features: 4
K: 1000, Fold Size: 1
Time for 1 feature: 9.43s
The accuracy after removing feature 1 and for features [2, 6, 8, 9] is 73.80%
The accuracy after removing feature 2 and for features [1, 6, 8, 9] is 90.50%
The accuracy after removing feature 6 and for features [1, 2, 8, 9] is 90.10%
The accuracy after removing feature 8 and for features [1, 2, 6, 9] is 81.30%
The accuracy after removing feature 9 and for features [1, 2, 6, 8] is 90.10%

Max accuracy of 90.50% without feature 2 and selected_features as [1, 6, 8, 9]
Average time for each feature: 9.95s
Total time for each feature: 49.73s

Total Instances: 1000, Total Features: 3
K: 1000, Fold Size: 1
Time for 1 feature: 8.87s
The accuracy after removing feature 1 and for features [6, 8, 9] is 73.30%
The accuracy after removing feature 6 and for features [1, 8, 9] is 93.90%
The accuracy after removing feature 8 and for features [1, 6, 9] is 83.10%
The accuracy after removing feature 9 and for features [1, 6, 8] is 92.40%

Max accuracy of 93.90% without feature 6 and selected_features as [1, 8, 9]
Average time for each feature: 10.26s
Total time for each feature: 41.06s

Total Instances: 1000, Total Features: 2
K: 1000, Fold Size: 1
Time for 1 feature: 10.10s
The accuracy after removing feature 1 and for features [8, 9] is 74.90%
The accuracy after removing feature 8 and for features [1, 9] is 84.70%
The accuracy after removing feature 9 and for features [1, 8] is 95.10%

Max accuracy of 95.10% without feature 9 and selected_features as [1, 8]
Average time for each feature: 9.81s
Total time for each feature: 29.43s
```

Figure 17: Continuation of the Traceback. Here, the sixth worst feature to remove is feature 2. The seventh worst feature to remove is Feature 6. The eighth worst feature to remove is Feature 9



```
Max accuracy of 95.10% without feature 9 and selected_features as [1, 8]
Average time for each feature: 9.81s
Total time for each feature: 29.43s

Total Instances: 1000, Total Features: 1
K: 1000, Fold Size: 1
Time for 1 feature: 10.20s
The accuracy after removing feature 1 and for features [8] is 71.60%
The accuracy after removing feature 8 and for features [1] is 83.50%

Max accuracy of 83.50% without feature 8 and selected_features as [1]
Average time for each feature: 10.17s
Total time for each feature: 20.34s

It took 8 mins and 58 secs to run.
----------------------------------------
Best Accuracy is 95.10% with features [1, 8]
```

Figure 18: Continuation of the Traceback. Here, the ninth worst feature to remove is feature 8. The final worst feature or the best feature is Feature 1.

8. Original Code

The original code can be found at

- The code is available on github → https://github.com/yashUcr773/CS_205_AI/tree/main/Projects/Project%202
- The code can also be run on google colab → **https://colab.research.google.com/drive/1VyzhBqGJSqQLU3EI7kodLMSerO-ZY5Us**

```
# -*- coding: utf-8 -*-
"""CS 205 AI Project 2.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1VyzhBqGJSqQLU3EI7kodLMSerO-ZY5Us

### Imports
"""

############################################################################
####################            IMPORTS           ####################
############################################################################

# To import and use dataset
import pandas as pd

# To work with pandas dataframe and plot graphs easily
import numpy as np

# To download the Dataset from cloud
from urllib import request

# To extract the dataset
import zipfile

# To time how long the execution takes place
import time

# To work with filesystem for extracting and locating the dataset
import os

# To plot the graphs
import matplotlib.pyplot as plt

# To get Real world dataset - Penguins Dataset
import seaborn as sns

# To convert categorical data into numerical data in real world dataset.
from sklearn.preprocessing import LabelEncoder
```

```
"""### Get and Setup Dataset"""

######################################################################
################            GET DATASETS            ###############
######################################################################
'''
Code the download the dataset.
The dataset is hosted on CDN so that it is available for everyone to download
'''

path_to_get_dataset_zip =
'https://d1u36hdvoy9y69.cloudfront.net/cs-205-ai/Project_2_synthetic_dataset/data_sets.zip'

# Hack to make it runnable in google colab as it does not support __file__
try:
    print(__file__)
except:
    __file__ = './content'

print(__file__)
base_path = os.path.dirname(os.path.abspath(__file__))+'/datasets'

# create the dataset directory if it does not exist
if not os.path.exists(base_path):
    os.makedirs(base_path)

# download the dataset and store it.
path_to_store_dataset_zip = f'{base_path}/data_set.zip'
request.urlretrieve(path_to_get_dataset_zip, path_to_store_dataset_zip)

######################################################################
################           UNZIP  DATASETS          ###############
######################################################################

# https://docs.python.org/3/library/zipfile.html

'''
Unzip the dataset into corresponding csv files
'''



def unzip_file(zip_path, extract_path):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)


unzip_file(path_to_store_dataset_zip, base_path)

######################################################################
################         PATH TO TEST DATASETS        ###############
######################################################################
```

```
'''
As I am solo and my DOB is 07/17/XXXX,
Select the datasets accordingly
'''

# day of month of smallest {07/17/XXXX} = 17
small_dataset_path = 'CS170_small_Data__17.txt'

# day of month of largest {07/17/XXXX} = 17
large_dataset_path = 'CS170_large_Data__17.txt'

# sum of months of both {07/17/XXXX} = 14
xxx_large_dataset_path = 'CS170_XXXlarge_Data__14.txt'

"""### Helper Functions"""

##########################################################################
################          HELPER FUNCTIONS          ###############
##########################################################################


def euclidean(x1, x2):
   '''
   Function to calculate euclidean distance between two points
   '''
   return np.sqrt(np.sum((x2 - x1)**2))


def print_formatted_time(time_input):
   '''
   funtion to take in seconds as input and
   print in Hours, minutes, and seconds
   '''
   hrs = int(time_input // 3600)
   mins = int((time_input % 3600) // 60)
   secs = int((time_input % 3600) % 60)
   if hrs:
      print(f'It took {hrs} hrs, {mins} mins and {secs} secs to run.')
   elif mins:
      print(f'It took {mins} mins and {secs} secs to run.')
   else:
      print(f'It took {secs} secs to run.')


def print_time(time_input):
   '''
   function to print the time with appropritate precision if between 0 and 1
   else print in HH, MM, SS format
   '''
   if time_input <= 1e-5:
      print(f'It took {time_input:.6f} secs to run.')
   elif time_input <= 1e-4:
        print(f'It took {time_input:.5f} secs to run.')
    elif time_input <= 1e-3:
        print(f'It took {time_input:.4f} secs to run.')
    elif time_input <= 1e-2:
        print(f'It took {time_input:.3f} secs to run.')
    elif time_input <= 1e-1:
        print(f'It took {time_input:.2f} secs to run.')
    elif time_input >= 0 and time_input <= 1:
        print(f'It took {time_input} secs to run.')
    else:
        print_formatted_time(time_input)

def add_line_break(lis_of_nums, break_every=2):
    '''
    function to split a list by adding \n to it.
    adds after every specified digit.
    Used for xticks values in graphs
    '''
    res = []
    for idx, i in enumerate(lis_of_nums):
        res.append(i)
        if idx != 0 and idx % break_every == 0:
            res.append('\n')
    return res

def plot_graphs(accuracy_map, algorithm = 'Forward Selection', dataset_size='Small'):
    '''
    Function to plot the feature graphs.
    Takes in accuracy_map array that is generated by the functions.
    '''
    plt.figure(1, figsize=(20, 5))
    plt.plot(accuracy_map[:, 0], accuracy_map[:, 1])
    plt.title(f'Accuracy vs Number of features for {algorithm} run on {dataset_size} Dataset')
    plt.xlabel('Selected features')
    plt.ylabel('Accuracy')
    plt.xticks(list(accuracy_map[:, 0]), list(
        [','.join(add_line_break([str(j) for j in i])) for i in accuracy_map[:, 2]]))
    plt.grid()
    plt.plot()
    plt.show()


def estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1):
    '''
    Function to check how many combinations of the Features the model needs to test and
    and how long will it take for the model to completely execute all the combinations
    '''

    df = pd.read_csv(f'{base_path}/{dataset_path}', sep=sep,
                header=None, engine='python')
    print(
        f'The dataset {dataset_path} has {df.shape[0]} instances with {df.shape[1] - 1} features')

    X = np.array(df[list(range(1, df.shape[1]))])
    Y = np.array(df[0])

    if validation_type not in ['LEAVE_ONE_OUT', 'K_FOLD_VALIDATION']:
        raise Exception('Not a valid validation')

    k_fold_k_value = k_val
    if validation_type == 'LEAVE_ONE_OUT':
        k_fold_k_value = df.shape[0]

    if sampling_factor < 0 or sampling_factor > 1:
        raise Exception('Not a valid sampling factor')

    if sampling == True:
        # shuffle the dataset
        indices = np.random.permutation(len(X))
        X = X[indices]
        Y = Y[indices]
        selected_len = int(len(indices) * sampling_factor)
        X = X[:selected_len]
        Y = Y[:selected_len]
        k_fold_k_value = X.shape[0]


    t0 = time.time()
    k_fold_acc = k_fold_cross_validation(
        X, Y, k_fold_k_value, best_so_far=-1, tolerence=0, verbose=False, prune_run=False)
    t1 = time.time()

    time_taken = t1 - t0
    total_combinations = (X.shape[1]*(X.shape[1]+1)) // 2

    print(
        f'K-fold cross validation accuracy on {dataset_path} for k = {k_fold_k_value} is with all features
selected is {k_fold_acc*100:.3f}%')
    print_time(time_taken)
    print()
    print(
        f'The model will run for {total_combinations} times and will take a total time of
{total_combinations*time_taken:.2f}s')
    print_time(total_combinations*time_taken)

"""### Main Functions"""

##########################################################################
################     NEAREST NEIGHBOR FUNCTION     ###############
##########################################################################


def knn(x_train, y_train, x_test, y_test):
    '''
    Function to find the accuracy of given train and test set using nearest neighbour algorithm
    '''
    correct = 0
    for i in range(0, x_test.shape[0]):
        distances = []

        for j in range(0, x_train.shape[0]):
            distances.append((y_train[j], euclidean(x_train[j], x_test[i])))

        distances = np.array(sorted(distances, key=lambda x: x[1]))
        y_pred = distances[0][0]

        if y_pred == y_test[i]:
            correct += 1

    accuracy = correct/x_test.shape[0]
    return accuracy

######################################################################
################          Cross Validation          ###############
######################################################################


def k_fold_cross_validation(X, Y, k, best_so_far, tolerence=5, verbose=False, prune_run=False):
    '''
    Function to apply k fold cross validation on a given dataset
    This function can be optimized to terminate runs if the running average is lower than the best_so_far
value provided.
    To terminate runs, pass the least required average, else pass in -1.
    The algorithm will run for a minimum of 100 iterations and then will tolerate atmost tolerence amount of
runs before terminating.
    '''

    accuracy_scores = []
    fold_size = len(X) // k

    # shuffle the dataset
    indices = np.random.permutation(len(X))
    X = X[indices]
    Y = Y[indices]

    if verbose:
        print(f'Total Instances: {X.shape[0]}, Total Features: {X.shape[1]}')
        print(f'K: {k}, Fold Size: {fold_size}')

    running_average = 0
    counter = 0

    for i in range(k):
        fold_start = i * fold_size
        fold_end = fold_start + fold_size
        # Create the training set by excluding the current fold
        X_train = np.concatenate((X[:fold_start], X[fold_end:]), axis=0)
        Y_train = np.concatenate((Y[:fold_start], Y[fold_end:]), axis=0)

        # Create the validation set from the current fold
        X_test = X[fold_start:fold_end]
        Y_test = Y[fold_start:fold_end]

        # Get Test Accuracy
        accuracy = knn(X_train, Y_train, X_test, Y_test)

        # Append to list
        accuracy_scores.append(accuracy)
        running_average = np.mean(accuracy_scores)

        # If the algorithm has run for 20 folds and the running average has not improved, then terminate the
run
        if prune_run == True and len(accuracy_scores) > 20 and best_so_far != -1 and running_average <
best_so_far:
            counter += 1
            if counter >= tolerence:
                return running_average

    # return average accuracy
    return np.mean(accuracy_scores)

#############################################################################
################        FORWARD SELECTION        ###############
#############################################################################


def forward_selection(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1, prune_run=False):

    df = pd.read_csv(f'{base_path}/{dataset_path}', sep=sep,
                     header=None, engine='python')
    print(
        f'The dataset {dataset_path} has {df.shape[0]} instances with {df.shape[1] - 1} features')

    X = np.array(df[list(range(1, df.shape[1]))])
    Y = np.array(df[0])

    if validation_type not in ['LEAVE_ONE_OUT', 'K_FOLD_VALIDATION']:
        raise Exception('Not a valid validation')

    k_fold_k_value = k_val
    if validation_type == 'LEAVE_ONE_OUT':
        k_fold_k_value = df.shape[0]

    if sampling_factor < 0 or sampling_factor > 1:
        raise Exception('Not a valid sampling factor')
    if sampling == True:
        # shuffle the dataset
        indices = np.random.permutation(len(X))
        X = X[indices]
        Y = Y[indices]
        selected_len = int(len(indices) * sampling_factor)
        X = X[:selected_len]
        Y = Y[:selected_len]
        k_fold_k_value = X.shape[0]


    values, counts = np.unique(Y, return_counts=True)
    default_rate = max(counts)/sum(counts)
    print(f'Default Rate is: {default_rate*100:.2f}% \n')

    selected_features = []
    accuracy_map = []
    accuracy_map.append((0, default_rate, []))
    best_so_far = default_rate
    best_features = []

    # run a loop from 0 to all features
    t00 = time.time()
    while len(selected_features) < df.shape[1] - 1:
        temp_acc_list = []
        time_per_feature = []
        verbose = True

        best_feature_accuracy = 0

        for i in range(1, df.shape[1]):
            if i in selected_features:
                continue

            new_features = list(selected_features)
            new_features.append(i)

            X = np.array(df[list(new_features)])
            Y = np.array(df[0])

            t0 = time.time()
            k_fold_acc = k_fold_cross_validation(
                X, Y, k_fold_k_value, best_feature_accuracy, tolerence=10, verbose=verbose,
prune_run=prune_run)
            t1 = time.time()
            if verbose == True:
                print(f'Time for 1 feature {t1 - t0:.2f}s')
            verbose = False

            print(
                f'The accuracy for features {new_features} is {k_fold_acc*100:.2f}%')
            if k_fold_acc > best_feature_accuracy:
                best_feature_accuracy = k_fold_acc

            temp_acc_list.append((i, k_fold_acc))
            time_per_feature.append(t1-t0)

        temp_acc_list = sorted(temp_acc_list, reverse=True, key=lambda x: x[1])

        selected_features.append(temp_acc_list[0][0])
        accuracy_map.append(
            (len(selected_features), temp_acc_list[0][1], list(selected_features)))

        print()
        print(
            f'Max accuracy of {temp_acc_list[0][1]*100:.2f}% for feature {temp_acc_list[0][0]} and the feature list
is {selected_features}')
        print(
            f'Average time for each feature: {(sum(time_per_feature)/len(time_per_feature)):.2f}s')
        print(f'Total time for each feature: {(sum(time_per_feature)):.2f}s')

        if(temp_acc_list[0][1] > best_so_far):
            best_so_far = temp_acc_list[0][1]
            best_features = list(selected_features)

        print()

    t11 = time.time()
    print_time(t11-t00)
    print('----------------------------------------')

    print(
        f'Best Accuracy is {best_so_far*100:.2f}% with features {best_features}')

    accuracy_map = np.array(accuracy_map,  dtype=object)
    return accuracy_map

######################################################################
################         BACKWARD ELIMINATION         ###############
######################################################################


def backward_elimination(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1, prune_run=False):

    df = pd.read_csv(f'{base_path}/{dataset_path}', sep=sep,
                     header=None, engine='python')
    print(
        f'The dataset {dataset_path} has {df.shape[0]} instances with {df.shape[1] - 1} features')

    X = np.array(df[list(range(1, df.shape[1]))])
    Y = np.array(df[0])
```

```
if validation_type not in ['LEAVE_ONE_OUT', 'K_FOLD_VALIDATION']:
    raise Exception('Not a valid validation')

k_fold_k_value = k_val
if validation_type == 'LEAVE_ONE_OUT':
    k_fold_k_value = df.shape[0]

if sampling_factor < 0 or sampling_factor > 1:
    raise Exception('Not a valid sampling factor')

if sampling == True:
    # shuffle the dataset
    indices = np.random.permutation(len(X))
    X = X[indices]
    Y = Y[indices]
    selected_len = int(len(indices) * sampling_factor)
    X = X[:selected_len]
    Y = Y[:selected_len]
    k_fold_k_value = X.shape[0]


values, counts = np.unique(Y, return_counts=True)
default_rate = max(counts)/sum(counts)
print(f'Default Rate is: {default_rate*100:.2f}% \n')

selected_features = list(range(1, df.shape[1]))  # Start with all features
accuracy_map = []
best_so_far = default_rate
best_features = selected_features.copy()

# get accuracy when all features are selected
k_fold_acc = k_fold_cross_validation(
    X, Y, k_fold_k_value, -1, tolerence=0, verbose=False, prune_run=False)
accuracy_map.append(
    (len(selected_features), k_fold_acc, list(selected_features)))

print(
    f'The accuracy for all features {selected_features} is {k_fold_acc*100:.2f}%')

t00 = time.time()
while len(selected_features) > 1:
    temp_acc_list = []
    time_per_feature = []
    verbose = True

    best_feature_accuracy = 0

    for i in selected_features:
        new_features = selected_features.copy()
        new_features.remove(i)
```

```
            X = np.array(df[list(new_features)])
            Y = np.array(df[0])

            t0 = time.time()
            k_fold_acc = k_fold_cross_validation(
                X, Y, k_fold_k_value, best_feature_accuracy, tolerence=10, verbose=verbose,
prune_run=prune_run)
            t1 = time.time()
            if verbose:
                print(f'Time for 1 feature: {t1 - t0:.2f}s')
            verbose = False

            print(
                f'The accuracy after removing feature {i} and for features {new_features} is
{k_fold_acc*100:.2f}%')

            if k_fold_acc > best_feature_accuracy:
                best_feature_accuracy = k_fold_acc

            temp_acc_list.append((i, k_fold_acc))
            time_per_feature.append(t1 - t0)

        temp_acc_list = sorted(temp_acc_list, reverse=True, key=lambda x: x[1])

        selected_features.remove(temp_acc_list[0][0])
        accuracy_map.append(
            (len(selected_features), temp_acc_list[0][1], list(selected_features)))

        print()
        print(
            f'Max accuracy of {temp_acc_list[0][1] * 100:.2f}% without feature {temp_acc_list[0][0]} and
selected_features as {selected_features}')
        print(
            f'Average time for each feature: {sum(time_per_feature) / len(time_per_feature):.2f}s')
        print(f'Total time for each feature: {sum(time_per_feature):.2f}s')
        print()

        if(temp_acc_list[0][1] > best_so_far):
            best_so_far = temp_acc_list[0][1]
            best_features = list(selected_features)

    t11 = time.time()
    print_time(t11 - t00)
    print('----------------------------------------')

    print(
        f'Best Accuracy is {best_so_far*100:.2f}% with features {best_features}')
    accuracy_map = np.array(accuracy_map,  dtype=object)
    return accuracy_map

"""### Test Data"""
```

```
#########################################################################
################        TESTING ON TEST DATA        ###############
#########################################################################

###################         TEST DATA 1          #################


def run_test_data_1(forward = True, backward = True):

   dataset_path = 'CS170_small_Data__32.txt'
   print('------------------ Time Estimation ---------------------')
   estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1)
   print()

   if forward:
      print('------------------ Forward Selection ----------------------')
      accuracy_map_forward_1 = forward_selection(
         dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
      plot_graphs(accuracy_map_forward_1)
      print()

   if backward:
      print('------------------ Backward Elimination ----------------------')
      accuracy_map_backward = backward_elimination(
         dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
      accuracy_map_updated_1 = np.array(accuracy_map_backward, dtype=object)
      accuracy_map_updated_1[:, 0] = len(
         accuracy_map_updated_1) - accuracy_map_updated_1[:, 0]
      plot_graphs(accuracy_map_updated_1)

###################         TEST DATA 2          ##################

def run_test_data_2(forward = True, backward = True):
   dataset_path = 'CS170_small_Data__33.txt'
   print('------------------ Time Estimation ----------------------')
   estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1)
   print()

   if forward:
      print('------------------ Forward Selection ----------------------')
      accuracy_map_forward_2 = forward_selection(
         dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
      plot_graphs(accuracy_map_forward_2)

      print()

   if backward:
      print('----------------- Backward Elimination ---------------------')
      accuracy_map_backward = backward_elimination(
         dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
```

```
        accuracy_map_updated_2 = np.array(accuracy_map_backward, dtype=object)
        accuracy_map_updated_2[:, 0] = len(
            accuracy_map_updated_2) - accuracy_map_updated_2[:, 0]
        plot_graphs(accuracy_map_updated_2)

##################          TEST DATA 3           ##################

def run_test_data_3(forward = True, backward = True):
    dataset_path = 'CS170_large_Data__33.txt'
    print('------------------ Time Estimation ----------------------')
    estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1)
    print()

    if forward:
        print('------------------ Forward Selection ----------------------')
        accuracy_map_forward_3 = forward_selection(
            dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
        plot_graphs(accuracy_map_forward_3)
        print()

    if backward:
        print('------------------ Backward Elimination ----------------------')
        accuracy_map_backward = backward_elimination(
            dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
        accuracy_map_updated_3 = np.array(accuracy_map_backward, dtype=object)
        accuracy_map_updated_3[:, 0] = len(
            accuracy_map_updated_3) - accuracy_map_updated_3[:, 0]
        plot_graphs(accuracy_map_updated_3)

##################          TEST DATA 4           ##################

def run_test_data_4(forward = True, backward = True):
    dataset_path = 'CS170_large_Data__33.txt'
    print('------------------ Time Estimation ----------------------')
    estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1)
    print()

    if forward:
        print('------------------ Forward Selection ----------------------')
        accuracy_map_forward_4 = forward_selection(
            dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
        plot_graphs(accuracy_map_forward_4)

        print()

    if backward:
        print('------------------ Backward Elimination ----------------------')
        accuracy_map_backward = backward_elimination(
            dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
        accuracy_map_updated_4 = np.array(accuracy_map_backward, dtype=object)
        accuracy_map_updated_4[:, 0] = len(
            accuracy_map_updated_4) - accuracy_map_updated_4[:, 0]
        plot_graphs(accuracy_map_updated_4)

"""### Selected Data"""

#####################################################################
################       Working on selected Data       ###############
#####################################################################

#################          SELECTED DATA 1           ###############
def run_selected_data_1(forward = True, backward = True):
    dataset_path = small_dataset_path
    estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1)
    print()

    if forward:
        accuracy_map_forward_5 = forward_selection(
            dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
        plot_graphs(accuracy_map_forward_5, 'Forward Selection', 'Small')
        print()

    if backward:
        accuracy_map_backward = backward_elimination(
            dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
        accuracy_map_updated_5 = np.array(accuracy_map_backward, dtype=object)
        accuracy_map_updated_5[:, 0] = len(
            accuracy_map_updated_5) - accuracy_map_updated_5[:, 0]
        plot_graphs(accuracy_map_updated_5, 'Backward Elimination', 'Small')

#################          SELECTED DATA 2           ###############
def run_selected_data_2(forward = True, backward = True):
    dataset_path = large_dataset_path
    estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT', k_val=2,
sampling=False, sampling_factor=1)
    print()

    if forward:
        accuracy_map_forward_6 = forward_selection(
            dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
        plot_graphs(accuracy_map_forward_6, 'Forward Selection', 'Large')
        print()

    if backward:
        accuracy_map_backward = backward_elimination(
            dataset_path, validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False, sampling_factor=1)
        accuracy_map_updated_6 = np.array(accuracy_map_backward, dtype=object)
        accuracy_map_updated_6[:, 0] = len(
            accuracy_map_updated_6) - accuracy_map_updated_6[:, 0]
        plot_graphs(accuracy_map_updated_6, 'Backward Elimination', 'Large')
```

```
#################          SELECTED DATA 3           ###############
def run_selected_data_3(forward = True, backward = True):

   dataset_path = xxx_large_dataset_path
   estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT',
               k_val=2, sampling=False, sampling_factor=1)
   print ()
   estimate_run_time(dataset_path, sep='  ', validation_type='K_FOLD_VALIDATION',
               k_val=2, sampling=False, sampling_factor=1)
   print ()
   estimate_run_time(dataset_path, sep='  ', validation_type='LEAVE_ONE_OUT',
               k_val=2, sampling=True, sampling_factor=0.3)
   print ()
   estimate_run_time(dataset_path, sep='  ', validation_type='K_FOLD_VALIDATION',
               k_val=2, sampling=True, sampling_factor=0.3)
   print ()

   if forward:
      accuracy_map_forward_7 = forward_selection(
         dataset_path, validation_type='K_FOLD_VALIDATION', k_val=2, sampling=True,
sampling_factor=0.3)
      plot_graphs(accuracy_map_forward_7, 'Forward Selection', 'XXXLarge')
      print()

   if backward:
      accuracy_map_backward = backward_elimination(
         dataset_path, validation_type='K_FOLD_VALIDATION', k_val=2, sampling=True,
sampling_factor=0.3)
      accuracy_map_updated_7 = np.array(accuracy_map_backward, dtype=object)
      accuracy_map_updated_7[:, 0] = len(
         accuracy_map_updated_7) - accuracy_map_updated_7[:, 0]
      plot_graphs(accuracy_map_updated_7, 'Backward Elimination', 'XXXLarge')

"""### UI"""

def main_block():
   print ('---- My Awesome Feature Search Algorithm ----\n')
   print ('Select the Dataset Size\n1. Small\n2. Large\n3. Extra Large')
   dataset_choice = int(input('Enter Choice.'))
   if dataset_choice not in [1, 2, 3]:
      os.system('cls')
      print('Please enter correct choice.\n')
      main_block()
      return

   print ('\nSelect the Algorithm\n1. Forward Selection\n2. Backward Elimination')
   algorithm_choice = int(input('Enter Choice.'))
   if algorithm_choice not in [1, 2]:
      os.system('cls')
      print('Please enter correct choice.\n')
      main_block()
      return
```

```
    datasets = [small_dataset_path, large_dataset_path, xxx_large_dataset_path]
    selected_dataset = datasets[dataset_choice-1]

    validation_type = 'LEAVE_ONE_OUT'
    k_val=2
    sampling=False
    sampling_factor=1

    if dataset_choice == 3:
        validation_type = 'K_FOLD_VALIDATION'
        k_val=2
        sampling=True
        sampling_factor=0.5

    if algorithm_choice == 1:
        estimate_run_time(selected_dataset, sep='  ', validation_type=validation_type, k_val=k_val,
sampling=sampling, sampling_factor=sampling_factor)
        accuracy_map_forward_6 = forward_selection(selected_dataset, validation_type=validation_type,
k_val=k_val, sampling=sampling, sampling_factor=sampling_factor)
        plot_graphs(accuracy_map_forward_6)
    else:
        estimate_run_time(selected_dataset, sep='  ', validation_type=validation_type, k_val=k_val,
sampling=sampling, sampling_factor=sampling_factor)
        accuracy_map_backward = backward_elimination(selected_dataset, validation_type=validation_type,
k_val=k_val, sampling=sampling, sampling_factor=sampling_factor)
        accuracy_map_updated_7 = np.array(accuracy_map_backward, dtype=object)
        accuracy_map_updated_7[:, 0] = len(accuracy_map_updated_7) - accuracy_map_updated_7[: ,0]
        plot_graphs(accuracy_map_updated_7)

"""### Real World Dataset"""

# https://github.com/allisonhorst/palmerpenguins

def real_world(forward = True, backward = True):
    real_world_path = 'realworld.csv'
    data = sns.load_dataset("penguins") # load penguins dataset from seaborn
    data = data.dropna() # drop samples with missing values (NaN)

    columns_to_number_map = {}
    number_to_columns_map = {}

    for idx, col in enumerate(data.columns):
        columns_to_number_map[col] = idx
        number_to_columns_map[idx] = col

    LE = LabelEncoder()
    data['species'] = LE.fit_transform(data['species'])
    data['island'] = LE.fit_transform(data['island'])
    data['sex'] = LE.fit_transform(data['sex'])

    data.to_csv(f'{base_path}/{real_world_path}',index=False, header=None,sep=' ')
    estimate_run_time(real_world_path, sep=' ', validation_type='LEAVE_ONE_OUT',
                k_val=2, sampling=False, sampling_factor=1)
    print()

    if forward:
        accuracy_map_forward_real = forward_selection(
            real_world_path, sep=' ', validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False,
sampling_factor=1, prune_run = False)
        print (number_to_columns_map)
        plot_graphs(accuracy_map_forward_real, 'Forward Selection', 'Real World')
        print()

    if backward:
        accuracy_map_backward = backward_elimination(
            real_world_path, sep=' ', validation_type='LEAVE_ONE_OUT', k_val=2, sampling=False,
sampling_factor=1, prune_run = False)
        accuracy_map_updated_real = np.array(accuracy_map_backward, dtype=object)
        accuracy_map_updated_real[:, 0] = len(
            accuracy_map_updated_real) - accuracy_map_updated_real[:, 0]
        print (number_to_columns_map)
        plot_graphs(accuracy_map_updated_real, 'Backward Elimination', 'Real World')

"""### All Function Calls"""

##########################################################################
####################       Function Calls        ###################
##########################################################################
# run_test_data_1(False, True)
# run_test_data_2(False, True)
# run_test_data_3(False, True)
# run_test_data_4(False, True)
# run_selected_data_1(False, True)
# run_selected_data_2(False, True)
# run_selected_data_3(False, True)
# real_world(False, True)
main_block()
```