**CS 205: Artificial Intelligence**

**Project 2**

**Name** : Yash Aggarwal

**SID**  : 862333037

**Email**  : [yagga004@ucr.edu](mailto:yagga004@ucr.edu)

**NetID**  : yagga004

**Date**  : 14/June/2023

In completing this assignment I consulted:

1. The Project Statement Handout provided.
   1. <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**](https://colab.research.google.com/drive/1VyzhBqGJSqQLU3EI7kodLMSerO-ZY5Us)

**Outline of this report**

1. Cover Page: (this page)
2. Report: Pages 02 - 11
3. Sample Trace of Forward Selection: Pages 12 - 16
4. Sample Trace of Backward Elimination: Pages 17 - 20
5. My Code: Page 21 onwards

**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.

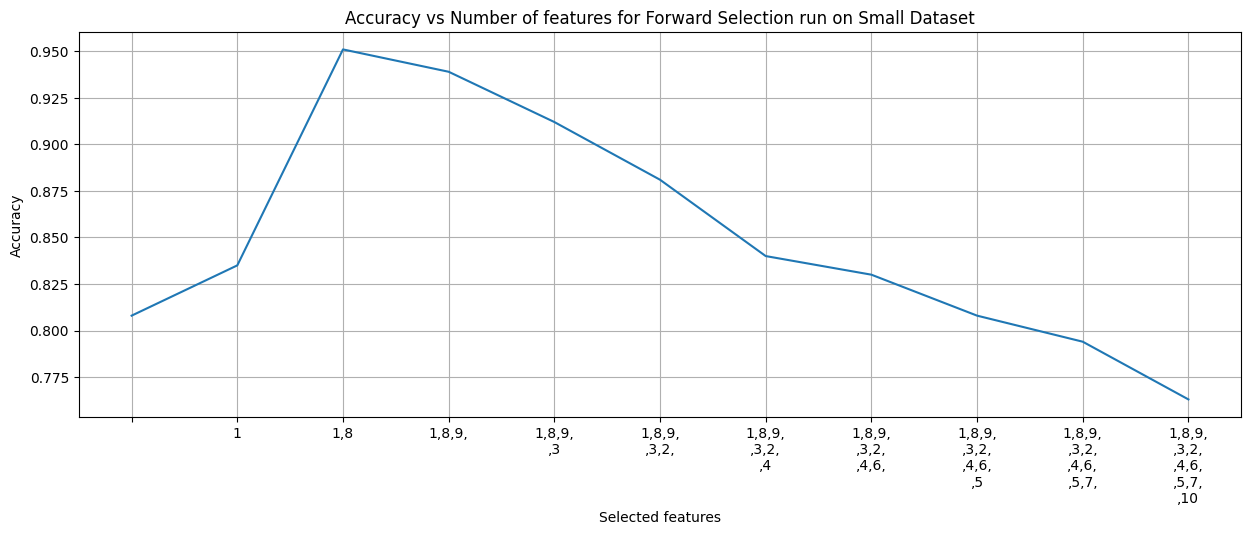


Figure 1. Accuracy for number of features selected in forward selection(scaled b/w 0-1)

### 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.

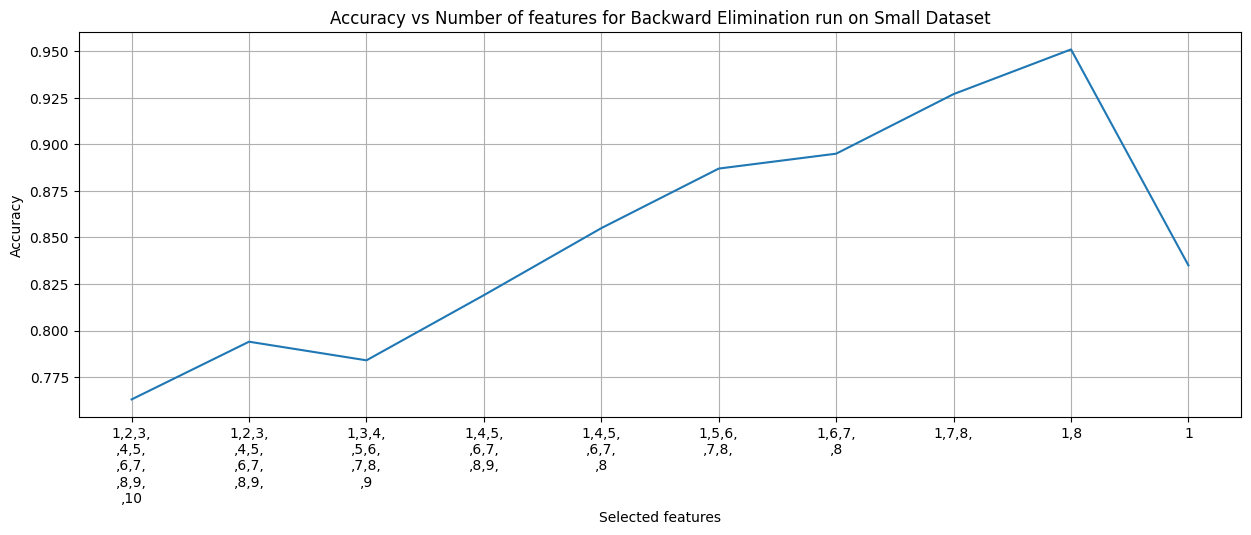


Figure 2. Accuracy for number of features selected in backward elimination(scaled b/w 0-1)

### 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.

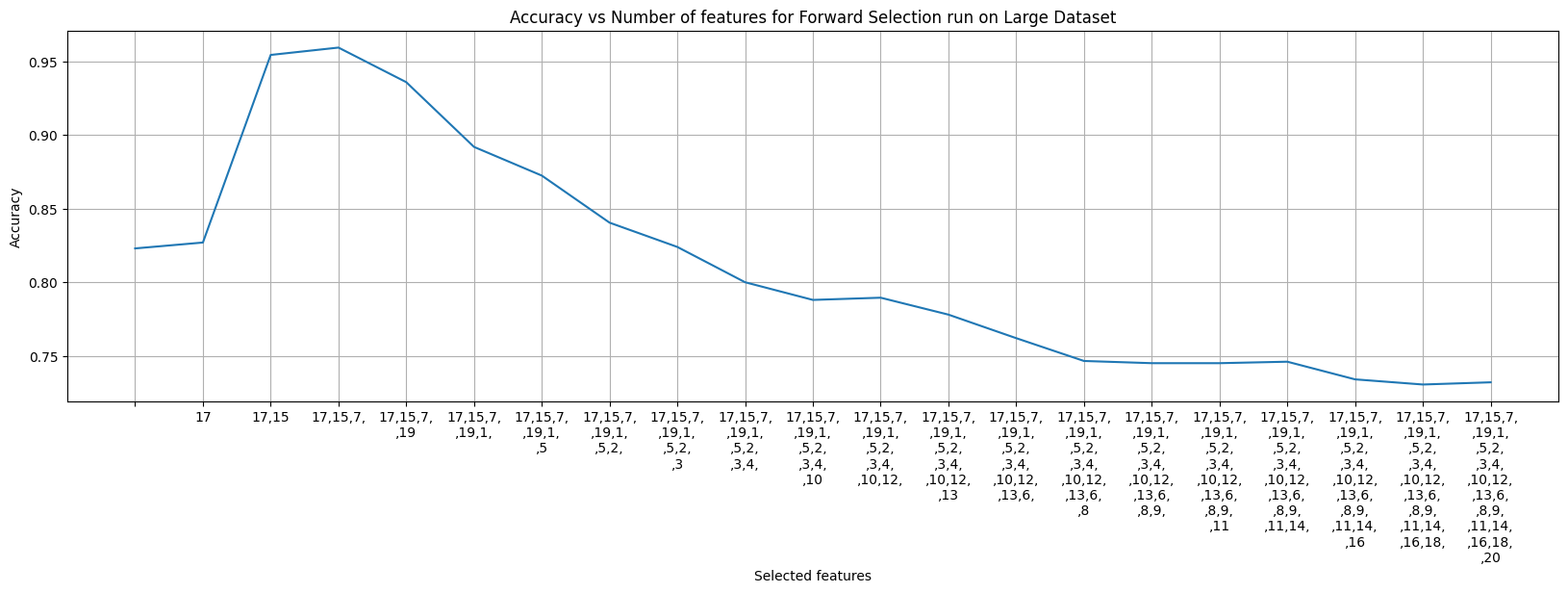


Figure 3. Accuracy for number of features selected in forward selection(scaled b/w 0-1)

### 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.

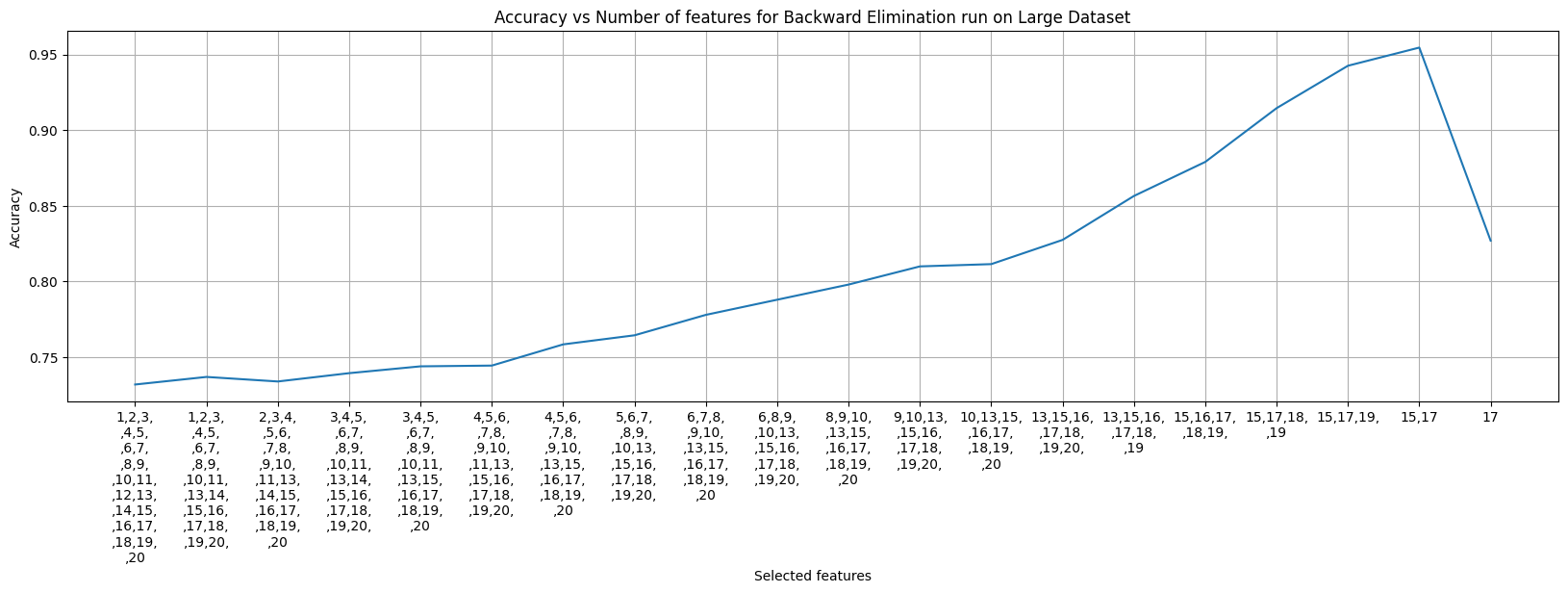


Figure 4. Accuracy for number of features selected in backward elimination(scaled b/w 0-1)

### 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.

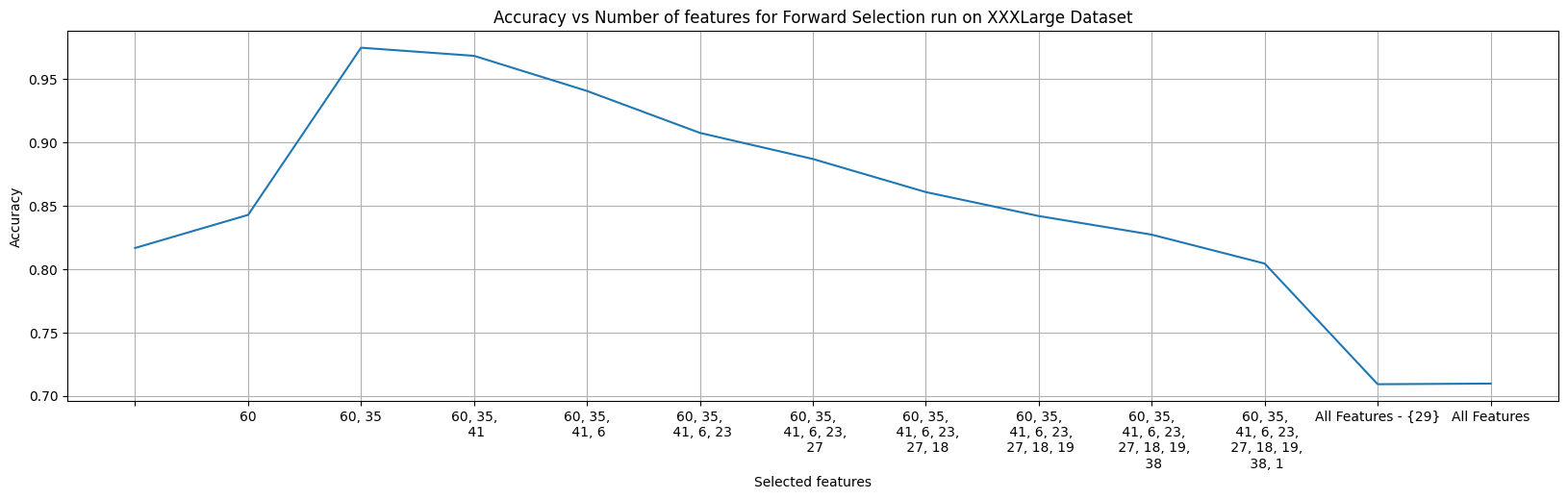


Figure 5. Accuracy for number of features selected in forward selection(scaled b/w 0-1)

### 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.

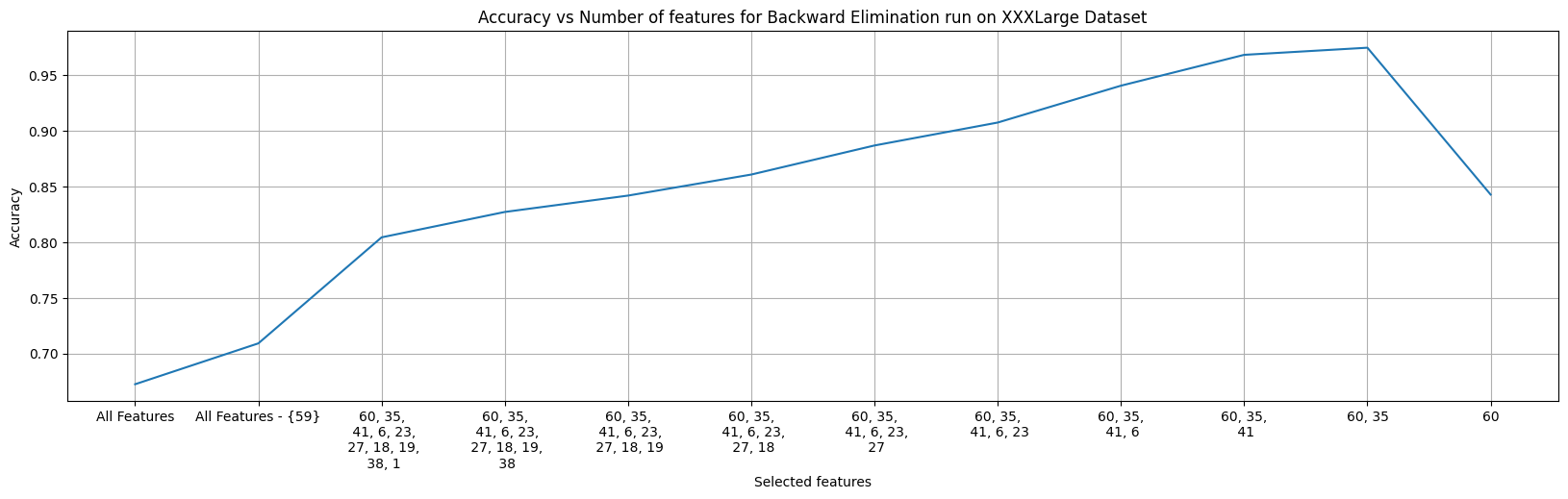


Figure 6. Accuracy for number of features selected in backward elimination(scaled b/w 0-1)

### 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.

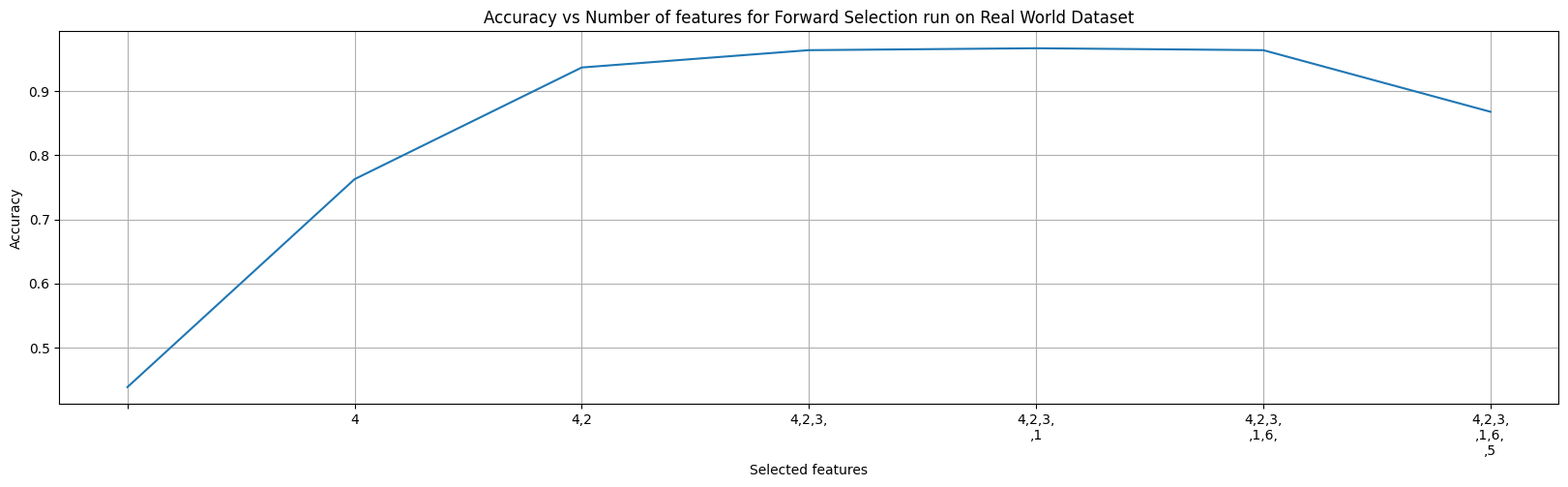


Figure 7. Accuracy for number of features selected in forward selection(scaled b/w 0-1)

### 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.

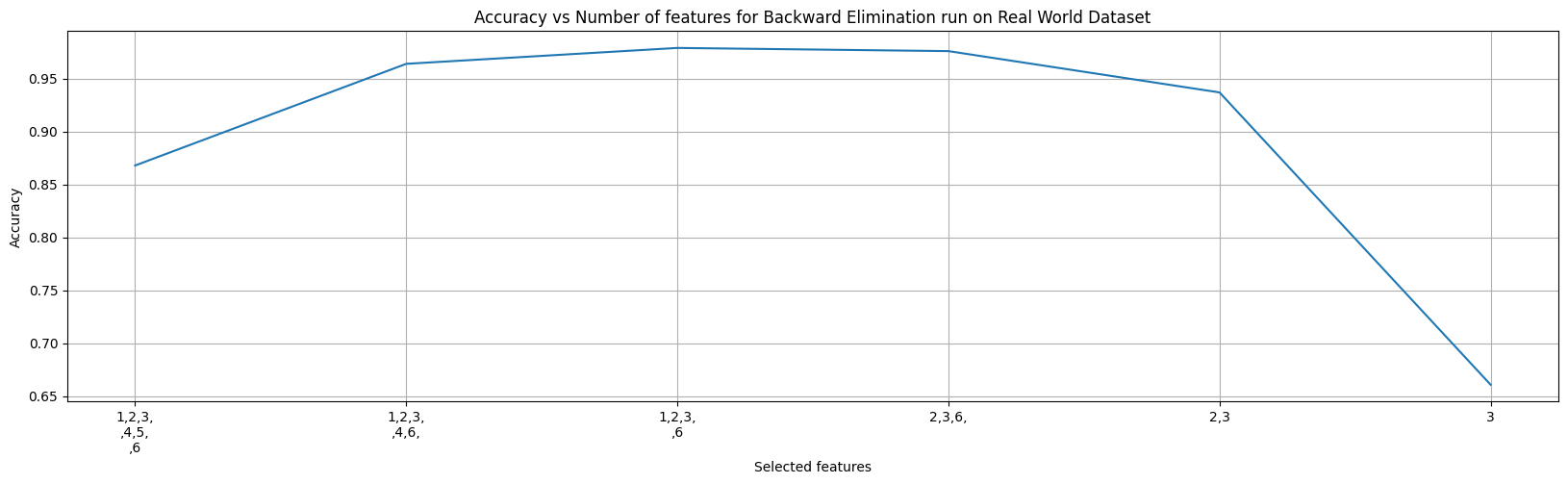


Figure 8. Accuracy for number of features selected in backward elimination(scaled b/w 0-1)

### 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.

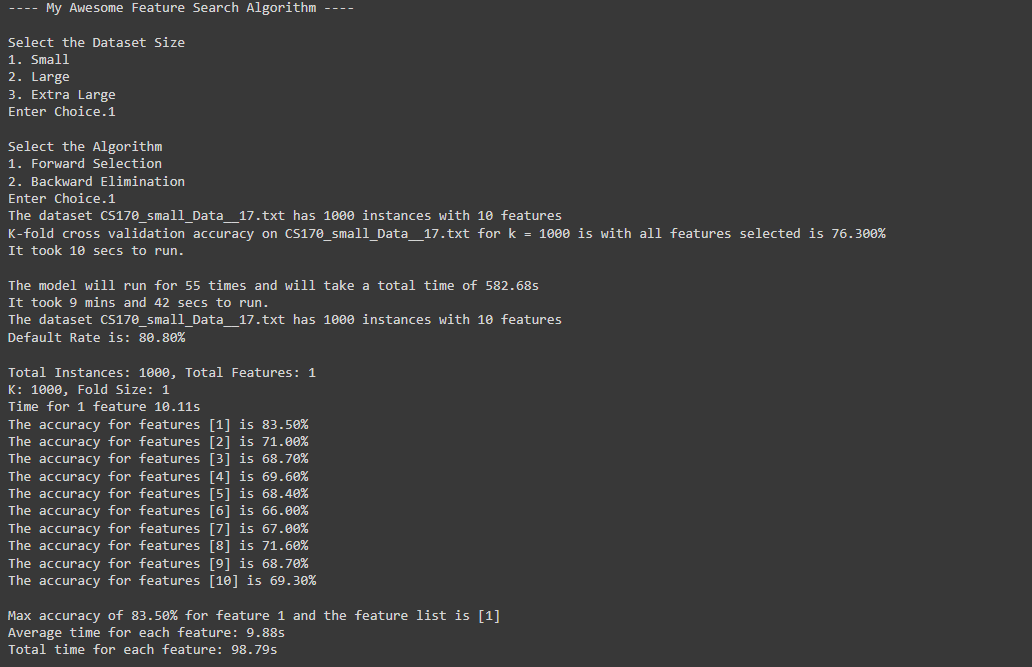
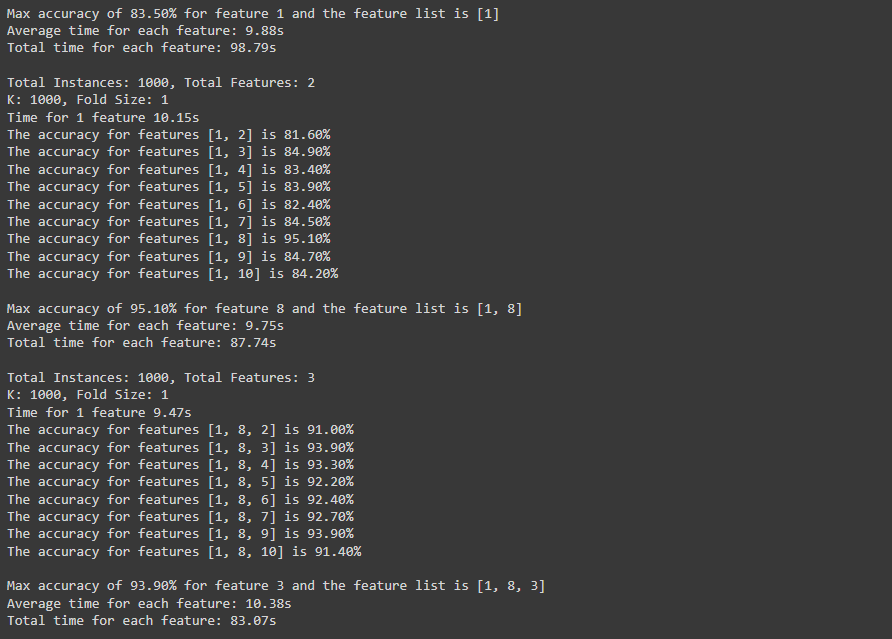
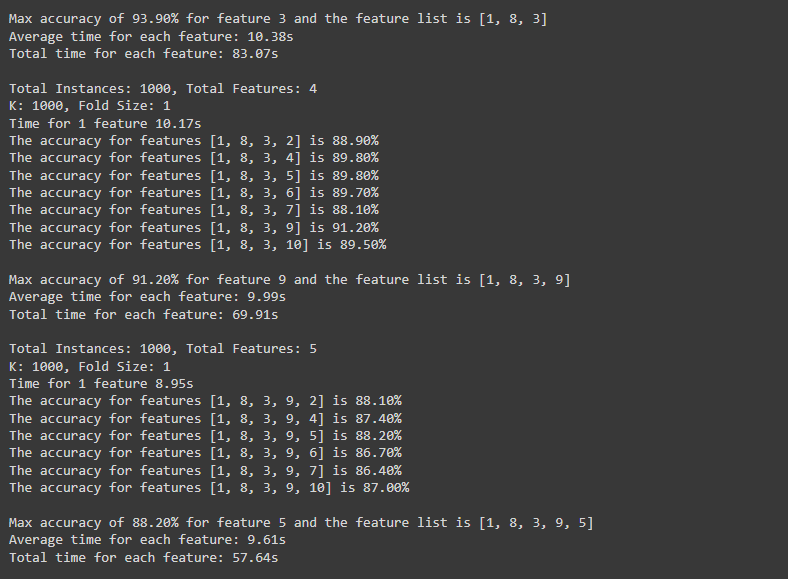
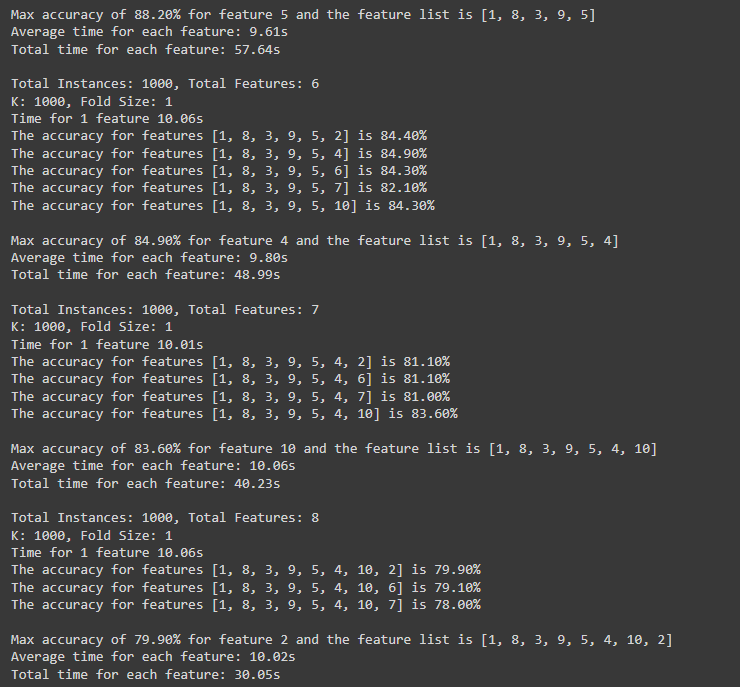
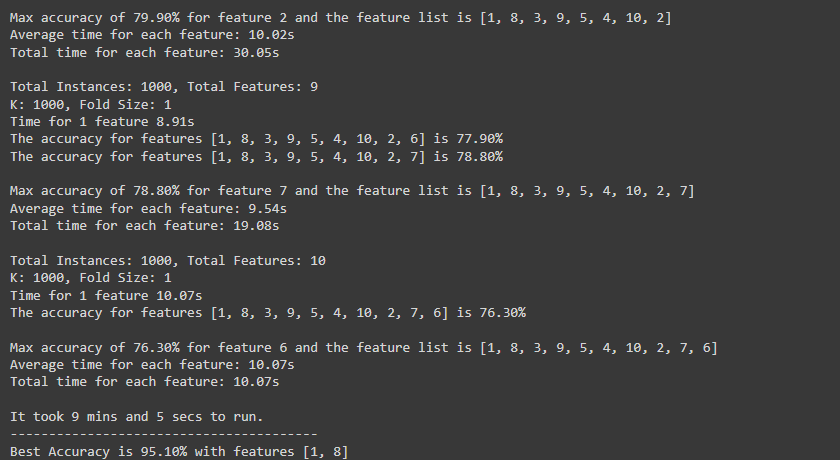
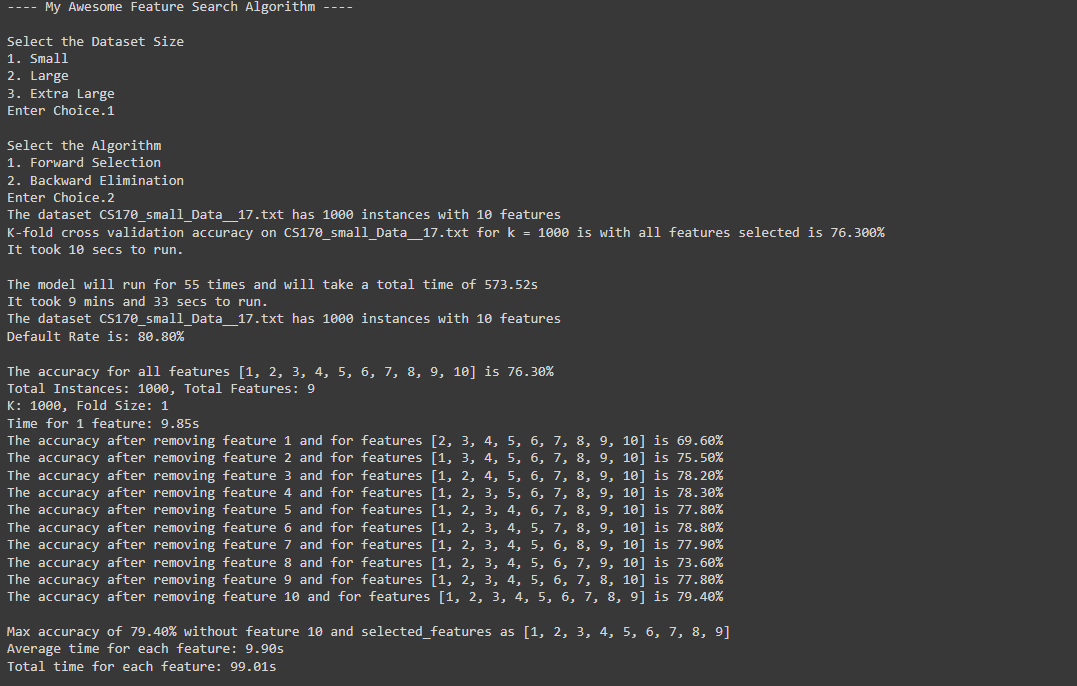
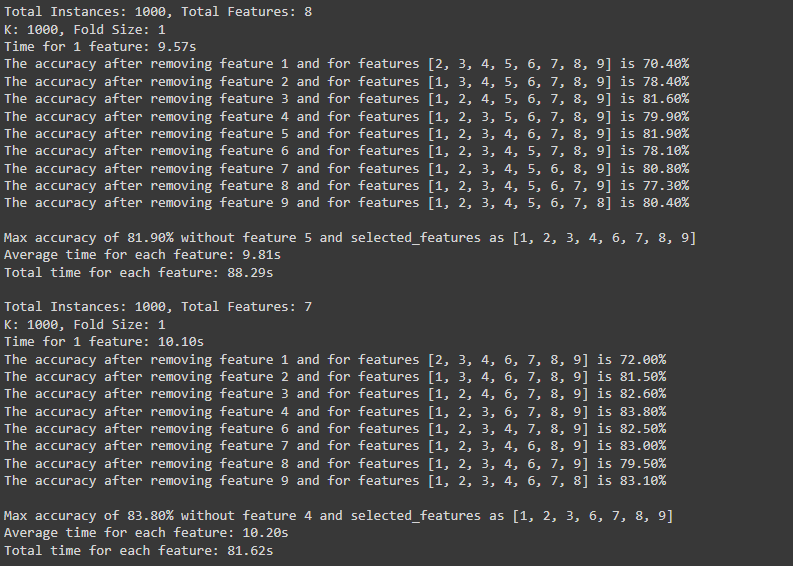


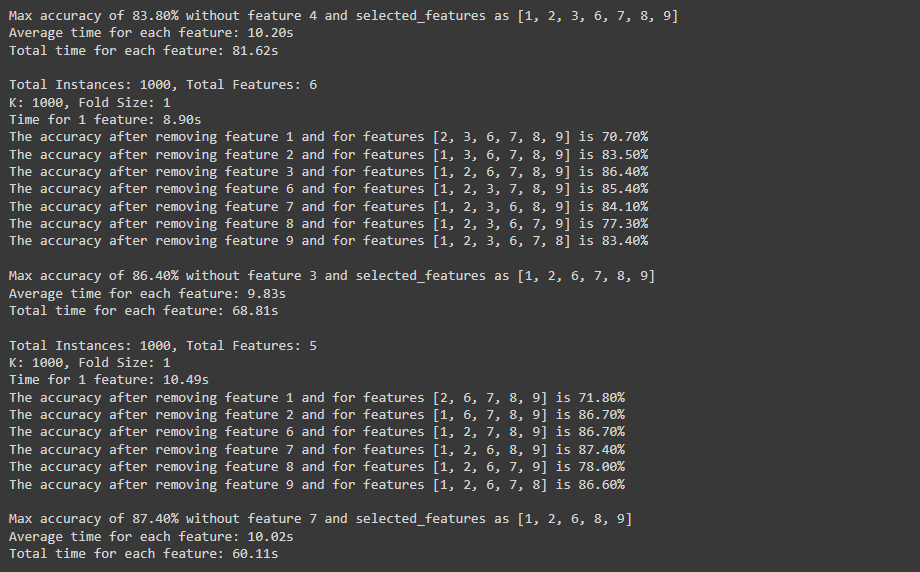
Figure 9: Start of Traceback on easy puzzle with forward selection algorithm. Here, the first best feature is feature 1.

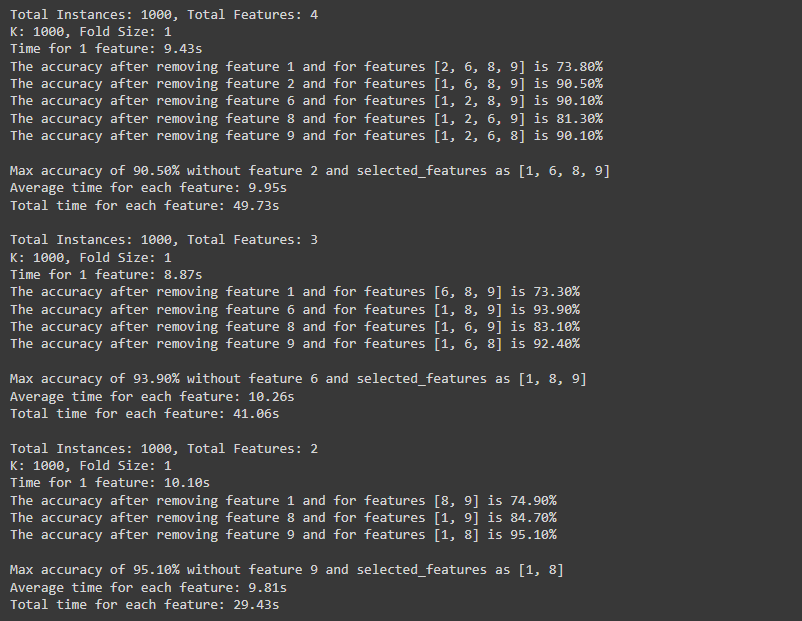
Figure 10: Continuation of the Traceback. Here, the second best feature is feature 8 and The third best feature is feature 3.Figure 11: Continuation of the Traceback. Here, the fourth best feature is feature 9 and The fifth best feature is feature 5.Figure 12: Continuation of the Traceback. Here, the sixth best feature is feature 4 and The seventh best feature is feature 10 and the eight best feature is feature 2.Figure 13: Continuation of the Traceback. Here, the ninth best feature is feature 7 and final feature is 6.

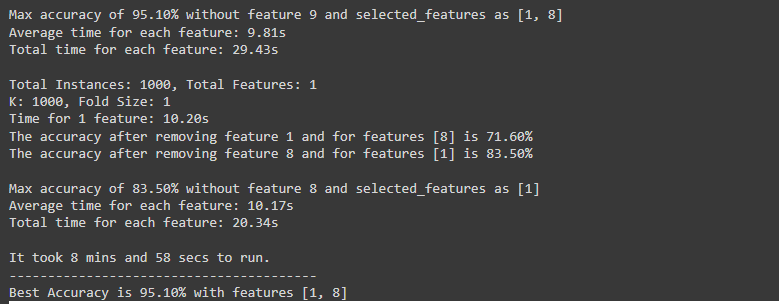
# 7. Traceback of Backward Elimination

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

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

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

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

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**](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()