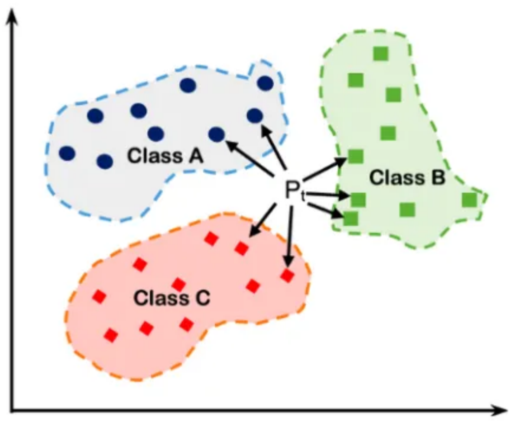


Feature Selection with Nearest Neighbor

Harrison Williams (hwill006)

`https://github.com/HWFord16/CS205_Project2`

CS205 - Introduction to Artificial Intelligence
University of California, Riverside

June 8th, 2025

Introduction

The second project for the Introduction to AI course taught by Dr.Eamonn Keogh is developing a program to perform feature selection using the K-Nearest Neighbor (KNN) classification. This technique was developed by Joe Hodges and Evelyn Fix[1] in 1951 while conducting research for the US Military.[2]. It is considered one of the most popular classification techniques in machine learning as it uses "proximity" of existing data points to make predictions on new individual data points [2]. This is accomplished by implementing two algorithms; Forward selection and Backward Elimination. The project's goal is to search for the best feature subset and calculate the highest accuracy from the provided small and large datasets with the techniques mentioned above. The datasets contain features and instances that the program must process. This implementation follows an object-oriented approach developed in Python. This report explores the design of the program, the K-NN classifier, the two algorithms, and a results evaluation of the overall execution of the program.

Overview of Algorithms

Forward Selection & Backward Elimination

These two algorithms are used with feature selection search algorithms as they are considered wrapper methods.[3] Wrapper methods are iterative greedy algorithms that assist in classification training and they search for combinations of features and compute the relation as well as accuracy between subsets of features.[3] Some of the pros and cons of these wrapper methods is that it can lead to better model performance and show the dependencies while the downsides are the computational overhead for processing large datasets.[3]. Forward selection starts with an empty set of features and keeps adding features after each iteration with the idea of improving predictive accuracy.[4]. Backward Elimination is the opposite approach where it starts with a full set of features and removes features to improve accuracy after each iteration.[4] The goal of both selection techniques is to assist in creating the best possible performing model.[3]

[1]Keogh, E. MachineLearning001. https://www.dropbox.com/scl/fo/lucvvfzc0fi7zf3tdlvpx/AJFYrHAXs3fO1nj_OeFxwvc?dl=0&e=21&preview=6__MachineLearning001.pptx&rlkey=cpr5hsj0grbd3pueqm05iao21

[2]What is k-Nearest Neighbor (kNN)? — A Comprehensive k-Nearest Neighbor Guide. (n.d.). Elastic. https://www.elastic.co/what-is/knn

[3]GeeksforGeeks. (2025, May 14). Feature selection techniques in machine learning. GeeksforGeeks. https://www.geeksforgeeks.org/feature-selection-techniques-in-machine-learning/#

[4]Keogh, E. MachineLearning002. https://www.dropbox.com/scl/fo/lucvvfzc0fi7zf3tdlvpx/AJFYrHAXs3fO1nj_OeFxwvc?dl=0&e=21&preview=7__MachineLearning002.pptx&rlkey=cpr5hsj0grbd3pueqm05iao21

Design

To complete the feature selection project using K-Nearest Neighbor (KNN) classification, the program is organized into five main Python files: main.py, Tree.py, Node.py, Classifier.py, and Validation.py. Main.py acts as the user interface, where users choose a dataset and then select either Forward Selection or Backward Elimination as the feature selection method. Based on these inputs, it initializes a Tree object and runs the corresponding algorithm. The core logic for feature selection is handled in Tree.py. This class manages the search process through a tree of feature subsets, starting from either an empty set (for forward selection) or the full set of features (for backward elimination). For each iteration, it evaluates different combinations of features to find the subset that gives the highest accuracy, using leave-one-out cross-validation (LOOCV) as the evaluation function. It also keeps a log of accuracy results for each subset size for later analysis to plot the results. Each feature subset is represented as a Node object, implemented in Node.py. A node tracks which features are currently selected and handles generating child nodes by either adding or removing features, depending on the algorithm. The KNN classifier itself is implemented in Classifier.py. It uses Euclidean distance to find the nearest neighbor for a new instance and return its label. This file also includes functions for reading dataset files and applying normalization to the feature values. Finally, Validation.py performs LOOCV. It splits the dataset so that each instance is tested once while the rest are used for training, and keeps track of how many predictions were correct. Overall, this program is designed to be modular with a clear separation between data handling, algorithm logic, and user interaction.

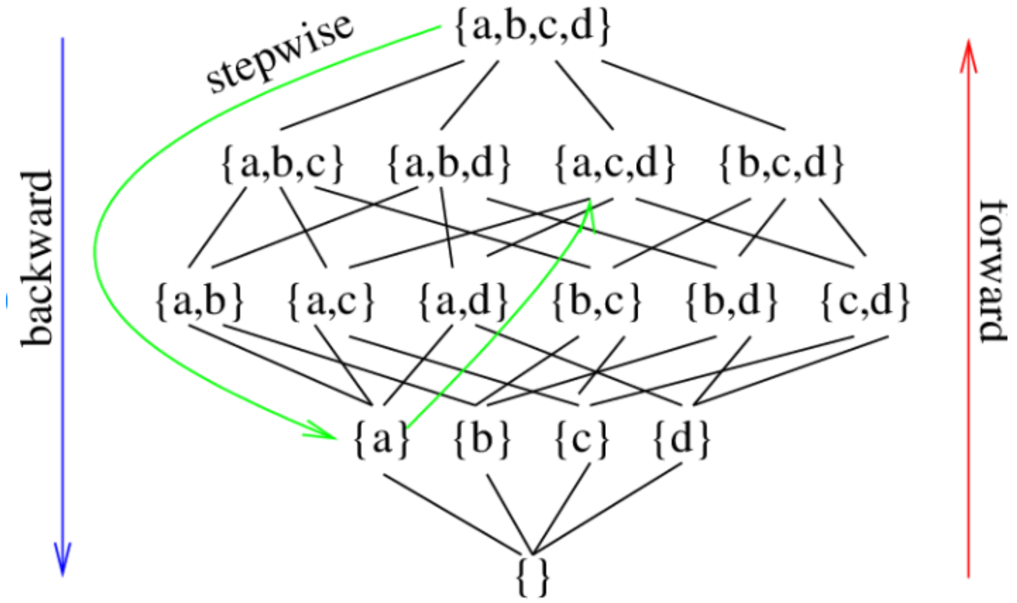


Figure 1: Wrapper methods visualization for KNN Classifier

Results

Forward Selection

The following plots were generated using a Python script running the MatPlotLib library and plotted the results from the forward selection algorithm for the large and small data sets. This plot displays the subsets of feature's indices with the highest accuracies during the search for each feature added. As one can see from both Fig. 2 and Fig. 3, they show the best feature subset was **{0,1}** out of 12 total features with an **97.2%** accuracy and **{25,46}** out of 50 features with an **96.7%** accuracy respectively.

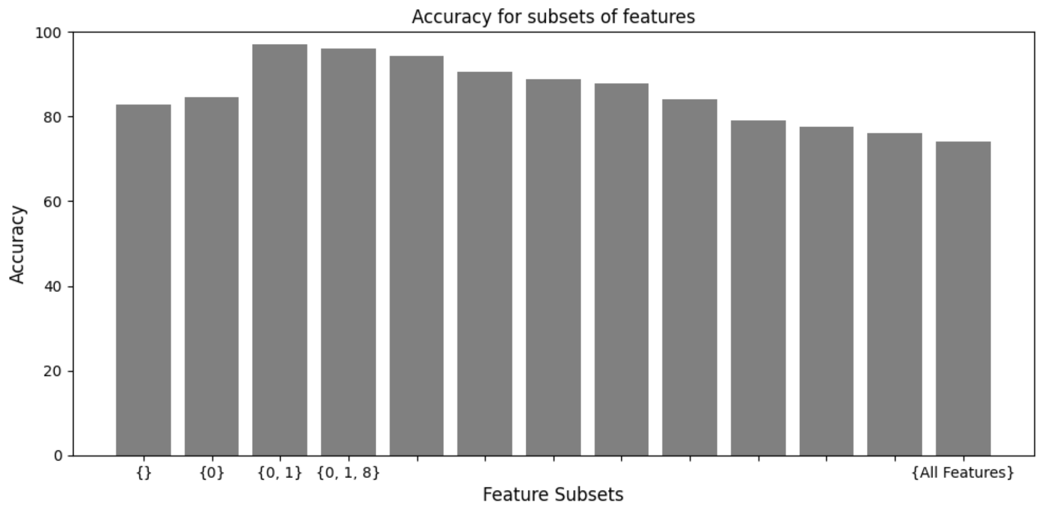


Figure 2: Generated from Small Data set #18 running forward selection.

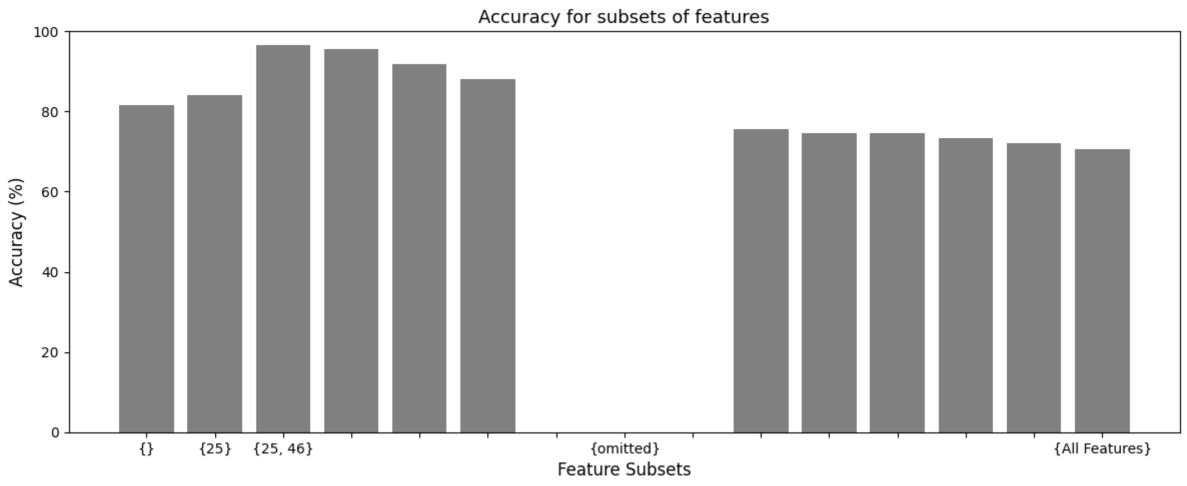


Figure 3: Generated from Large Data set #10 running forward selection.

Backward Elimination

The following plots plotted the results from the backward elimination algorithm for the large and small data sets. This plot displays the subsets of feature's indices with the highest accuracies during the search for when a feature is removed . As one can see from both Fig. 4 and Fig. 5, they show the best feature subset was **{0,1}** out of 12 total features with an **97.2%** accuracy and **{25,37}** out of 50 features with an **84.5%** accuracy respectively.

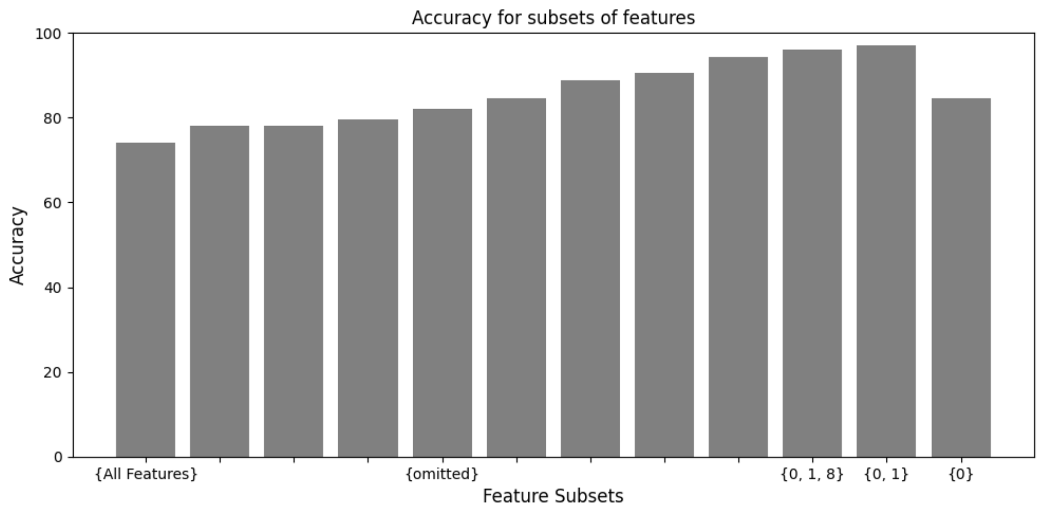


Figure 4: Generated from Small Data set #18 running backward elimination.

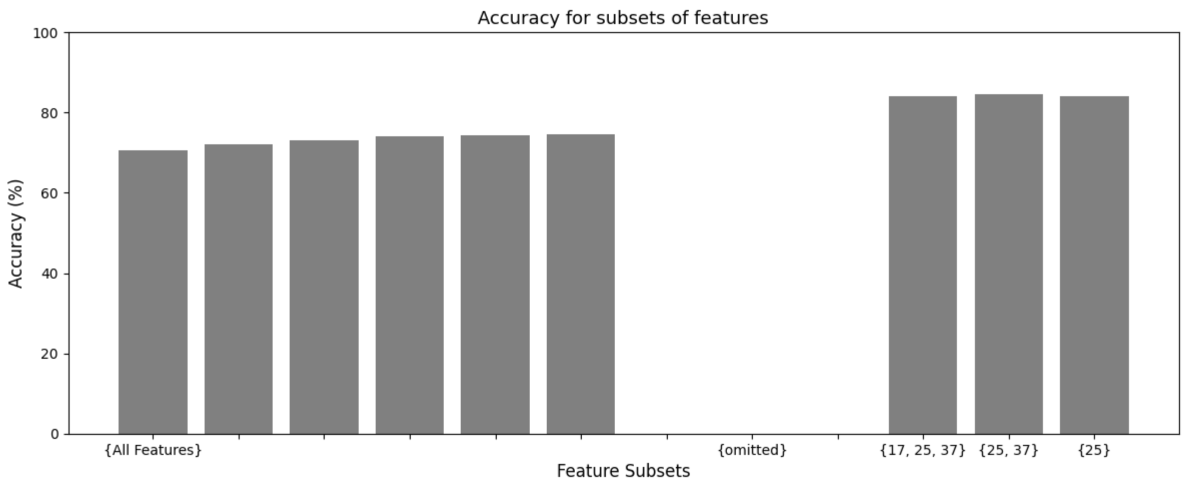


Figure 5: Generated from Large Data set #10 running backward elimination.

Performance

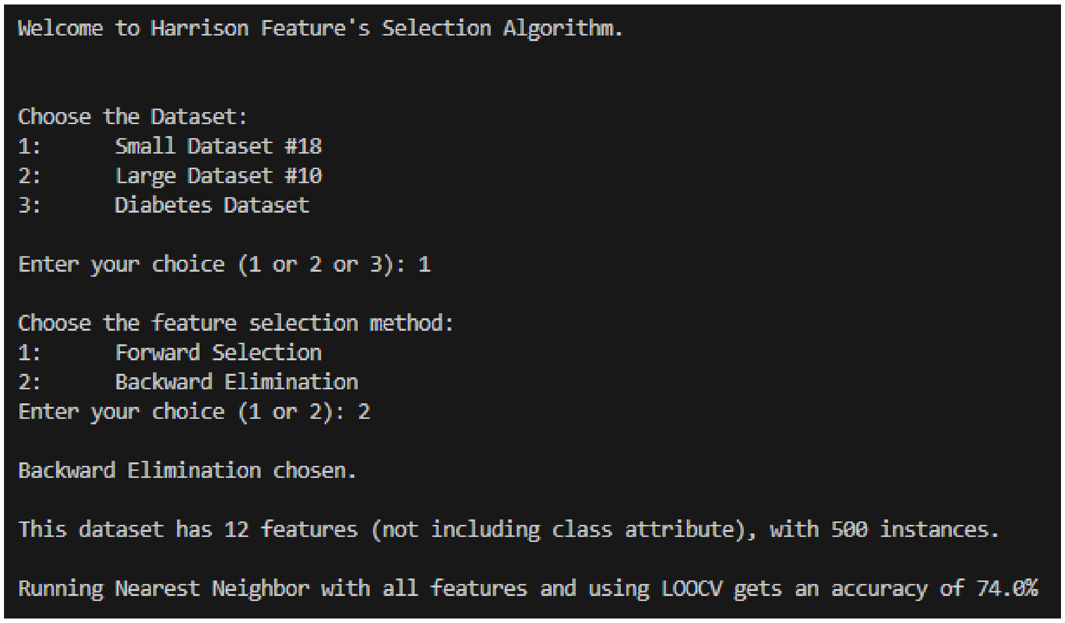
The following project was developed and deployed on a Windows 10 desktop that contains the AMD Ryzen 7 3700x which contains 8-cores and 32GB RAM. The program processed both datasets Small #18 with 12 features and 500 instances and Large #10 with 50 features and 1000 instances and the following table displays the results of runtime for both selection algorithms on both datasets.

	Small Dataset #18	**Large Dataset #10**
Forward Selection	0.65 s	43.58 s
Backward Elimination	0.73 s	78.24 s

Table 1: K-NN Classification Runtimes with both algorithms

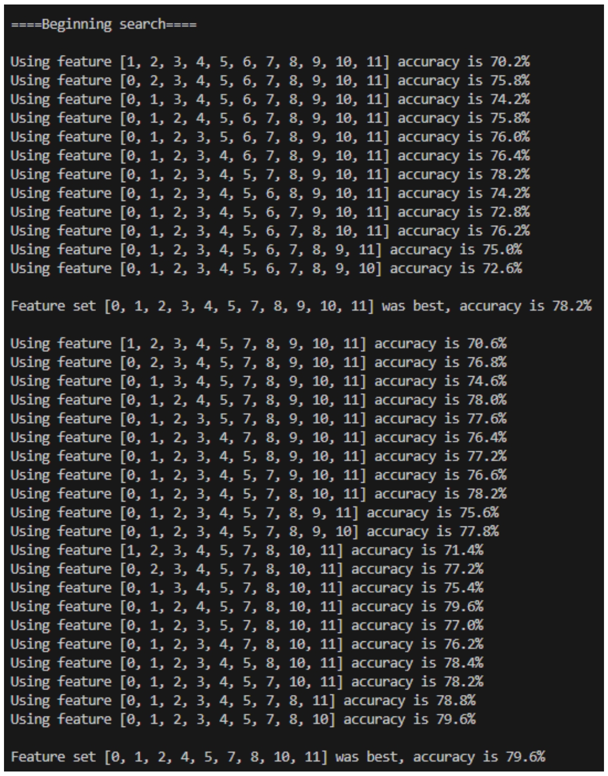
Trace

The following is the trace for the Backward Elimination algorithm executed on the small data set seen in Fig. 6 The forward selection on the small data set looks similar to this except it starts with an empty set and the traces for the large dataset are way too long to be displayed in the report. Please refer to the GitHub link above to run the program and view the traces for the large dataset.

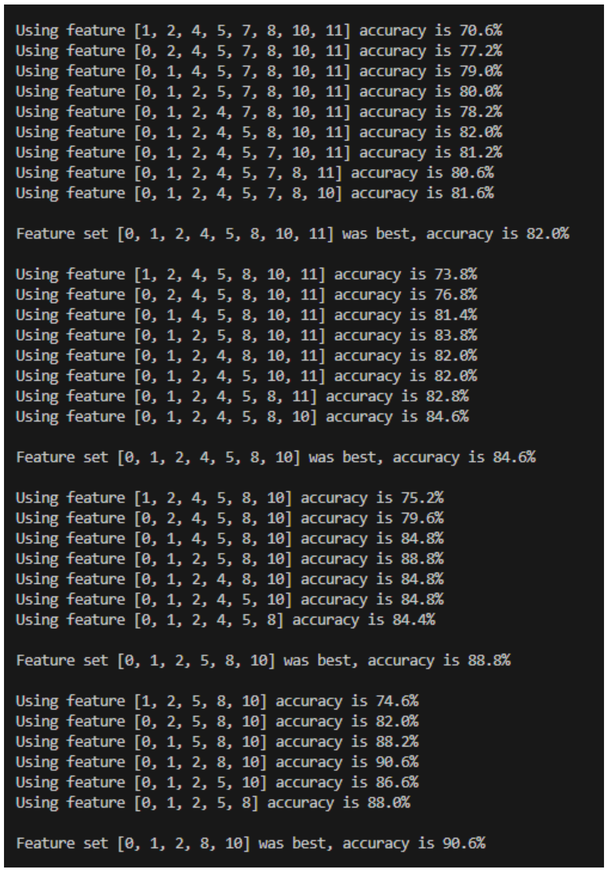


```
Welcome to Harrison Feature's Selection Algorithm.


Choose the Dataset:
1:      Small Dataset #18
2:      Large Dataset #10
3:      Diabetes Dataset

Enter your choice (1 or 2 or 3): 1

Choose the feature selection method:
1:      Forward Selection
2:      Backward Elimination
Enter your choice (1 or 2): 2

Backward Elimination chosen.

This dataset has 12 features (not including class attribute), with 500 instances.

Running Nearest Neighbor with all features and using LOOCV gets an accuracy of 74.0%
```

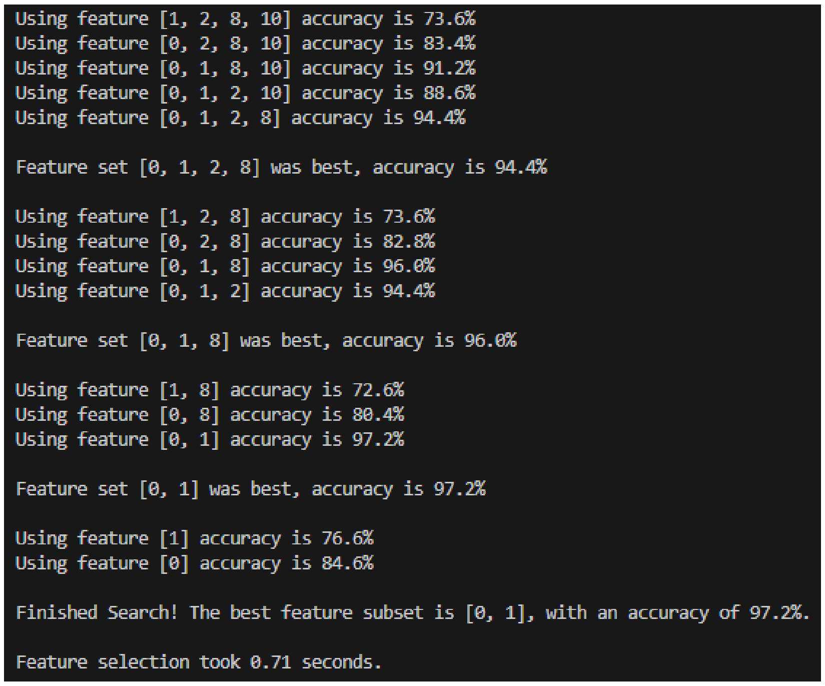
Figure 6: Trace output of Backward Elimination for small dataset #18.



```
====Beginning search====

Using feature [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] accuracy is 70.2%
Using feature [0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] accuracy is 75.8%
Using feature [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11] accuracy is 74.2%
Using feature [0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11] accuracy is 75.8%
Using feature [0, 1, 2, 3, 5, 6, 7, 8, 9, 10, 11] accuracy is 76.0%
Using feature [0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11] accuracy is 76.4%
Using feature [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11] accuracy is 78.2%
Using feature [0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11] accuracy is 74.2%
Using feature [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11] accuracy is 72.8%
Using feature [0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11] accuracy is 76.2%
Using feature [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11] accuracy is 75.0%
Using feature [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is 72.6%

Feature set [0, 1, 2, 3, 4, 5, 7, 8, 9, 10, 11] was best, accuracy is 78.2%

Using feature [1, 2, 3, 4, 5, 7, 8, 9, 10, 11] accuracy is 70.6%
Using feature [0, 2, 3, 4, 5, 7, 8, 9, 10, 11] accuracy is 76.8%
Using feature [0, 1, 3, 4, 5, 7, 8, 9, 10, 11] accuracy is 74.6%
Using feature [0, 1, 2, 4, 5, 7, 8, 9, 10, 11] accuracy is 78.0%
Using feature [0, 1, 2, 3, 5, 7, 8, 9, 10, 11] accuracy is 77.6%
Using feature [0, 1, 2, 3, 4, 7, 8, 9, 10, 11] accuracy is 76.4%
Using feature [0, 1, 2, 3, 4, 5, 8, 9, 10, 11] accuracy is 77.2%
Using feature [0, 1, 2, 3, 4, 5, 7, 9, 10, 11] accuracy is 76.6%
Using feature [0, 1, 2, 3, 4, 5, 7, 8, 10, 11] accuracy is 78.2%
Using feature [0, 1, 2, 3, 4, 5, 7, 8, 9, 11] accuracy is 75.6%
Using feature [0, 1, 2, 3, 4, 5, 7, 8, 9, 10] accuracy is 77.8%
Using feature [1, 2, 3, 4, 5, 7, 8, 10, 11] accuracy is 71.4%
Using feature [0, 2, 3, 4, 5, 7, 8, 10, 11] accuracy is 77.2%
Using feature [0, 1, 3, 4, 5, 7, 8, 10, 11] accuracy is 75.4%
Using feature [0, 1, 2, 4, 5, 7, 8, 10, 11] accuracy is 79.6%
Using feature [0, 1, 2, 3, 5, 7, 8, 10, 11] accuracy is 77.0%
Using feature [0, 1, 2, 3, 4, 7, 8, 10, 11] accuracy is 76.2%
Using feature [0, 1, 2, 3, 4, 5, 8, 10, 11] accuracy is 78.4%
Using feature [0, 1, 2, 3, 4, 5, 7, 10, 11] accuracy is 78.2%
Using feature [0, 1, 2, 3, 4, 5, 7, 8, 11] accuracy is 78.8%
Using feature [0, 1, 2, 3, 4, 5, 7, 8, 10] accuracy is 79.6%

Feature set [0, 1, 2, 4, 5, 7, 8, 10, 11] was best, accuracy is 79.6%
```



```
Using feature [1, 2, 4, 5, 7, 8, 10, 11] accuracy is 70.6%
Using feature [0, 2, 4, 5, 7, 8, 10, 11] accuracy is 77.2%
Using feature [0, 1, 4, 5, 7, 8, 10, 11] accuracy is 79.0%
Using feature [0, 1, 2, 5, 7, 8, 10, 11] accuracy is 80.0%
Using feature [0, 1, 2, 4, 7, 8, 10, 11] accuracy is 78.2%
Using feature [0, 1, 2, 4, 5, 8, 10, 11] accuracy is 82.0%
Using feature [0, 1, 2, 4, 5, 7, 10, 11] accuracy is 81.2%
Using feature [0, 1, 2, 4, 5, 7, 8, 11] accuracy is 80.6%
Using feature [0, 1, 2, 4, 5, 7, 8, 10] accuracy is 81.6%

Feature set [0, 1, 2, 4, 5, 8, 10, 11] was best, accuracy is 82.0%

Using feature [1, 2, 4, 5, 8, 10, 11] accuracy is 73.8%
Using feature [0, 2, 4, 5, 8, 10, 11] accuracy is 76.8%
Using feature [0, 1, 4, 5, 8, 10, 11] accuracy is 81.4%
Using feature [0, 1, 2, 5, 8, 10, 11] accuracy is 83.8%
Using feature [0, 1, 2, 4, 8, 10, 11] accuracy is 82.0%
Using feature [0, 1, 2, 4, 5, 10, 11] accuracy is 82.0%
Using feature [0, 1, 2, 4, 5, 8, 11] accuracy is 82.8%
Using feature [0, 1, 2, 4, 5, 8, 10] accuracy is 84.6%

Feature set [0, 1, 2, 4, 5, 8, 10] was best, accuracy is 84.6%

Using feature [1, 2, 4, 5, 8, 10] accuracy is 75.2%
Using feature [0, 2, 4, 5, 8, 10] accuracy is 79.6%
Using feature [0, 1, 4, 5, 8, 10] accuracy is 84.8%
Using feature [0, 1, 2, 5, 8, 10] accuracy is 88.8%
Using feature [0, 1, 2, 4, 8, 10] accuracy is 84.8%
Using feature [0, 1, 2, 4, 5, 10] accuracy is 84.8%
Using feature [0, 1, 2, 4, 5, 8] accuracy is 84.4%

Feature set [0, 1, 2, 5, 8, 10] was best, accuracy is 88.8%

Using feature [1, 2, 5, 8, 10] accuracy is 74.6%
Using feature [0, 2, 5, 8, 10] accuracy is 82.0%
Using feature [0, 1, 5, 8, 10] accuracy is 88.2%
Using feature [0, 1, 2, 8, 10] accuracy is 90.6%
Using feature [0, 1, 2, 5, 10] accuracy is 86.6%
Using feature [0, 1, 2, 5, 8] accuracy is 88.0%

Feature set [0, 1, 2, 8, 10] was best, accuracy is 90.6%
```



```
Using feature [1, 2, 8, 10] accuracy is 73.6%
Using feature [0, 2, 8, 10] accuracy is 83.4%
Using feature [0, 1, 8, 10] accuracy is 91.2%
Using feature [0, 1, 2, 10] accuracy is 88.6%
Using feature [0, 1, 2, 8] accuracy is 94.4%

Feature set [0, 1, 2, 8] was best, accuracy is 94.4%

Using feature [1, 2, 8] accuracy is 73.6%
Using feature [0, 2, 8] accuracy is 82.8%
Using feature [0, 1, 8] accuracy is 96.0%
Using feature [0, 1, 2] accuracy is 94.4%

Feature set [0, 1, 8] was best, accuracy is 96.0%

Using feature [1, 8] accuracy is 72.6%
Using feature [0, 8] accuracy is 80.4%
Using feature [0, 1] accuracy is 97.2%

Feature set [0, 1] was best, accuracy is 97.2%

Using feature [1] accuracy is 76.6%
Using feature [0] accuracy is 84.6%

Finished Search! The best feature subset is [0, 1], with an accuracy of 97.2%.

Feature selection took 0.71 seconds.
```

Part 2

Results

This part was to run our K-NN algorithm developed and tested in Part 1 and apply it to a real world data set. The following data set we will apply our project to is the ”Early Stage Diabetes Risk Prediction” which has **16** features and **520** instances. This dataset was referenced from UC Irvine's Machine Learning Repository.[5] This data set had no missing values and was primarily integer data (binary values) and one categorical feature which was gender. Data pre-processing was completed to convert this categorical feature to a binary one where $Male \longrightarrow 1$ and $Female \longrightarrow 0$. After that was completed, the dataset was now ready to be used with the program running the Forward Selection Algorithm.

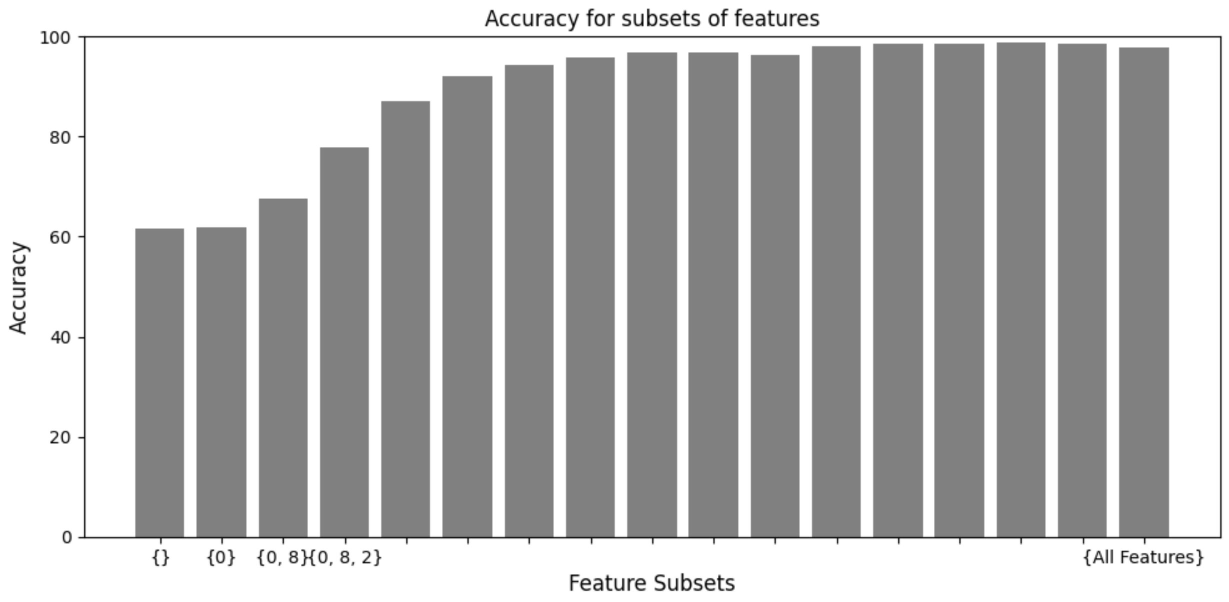


Figure 7: Plot of diabetes risk data from UCI ML Repo after running Forward Selection.

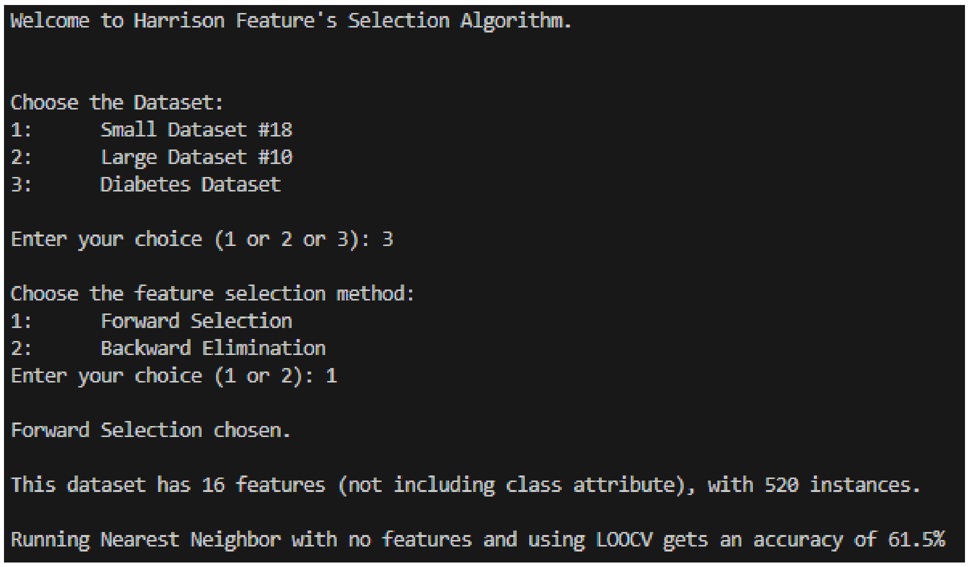
	Early Stage Diabetes Risk Prediction Dataset
Runtime	0.65 s
Best Feature Subset	{0, 8, 2, 5, 4, 9, 13, 11, 6, 1, 3, 7, 15, 14}
Accuracy	98.8%

Table 2: Diabetes Data K-NN classification algorithm results.

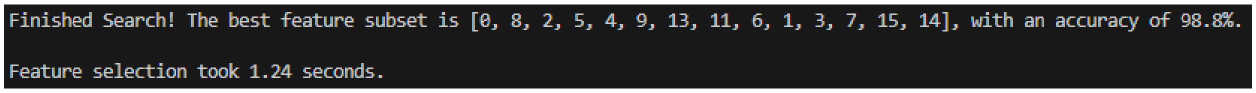
[5]UCI Machine Learning Repository. `https://archive.ics.uci.edu/dataset/529/early+stage+diabetes+risk+prediction+dataset`

Takeaway

As one can see from Fig. 8, and Table 2, diabetes prediction is quite accurate when the subset of features contains as many features as possible to indicate an early risk of diabetes. The features are the following: {Age, Gender, Polyuria , Polydipsia, weight loss, weakness, Polyphagia, Genital thrush, visual blurring, Itching, Irritability, delayed healing, partial paresis, muscle stiffness, Alopecia, Obesity} with the classes being a binary {positive, negative} for potential diagnosis of diabetes. So based on the results, it makes sense that if as many features are marked as the binary value of 1 which means "True" for the feature being present for a patient, then the likelihood for that patient to be classified as having diabetes is quite high and can be predicted with high accuracy.



```
Welcome to Harrison Feature's Selection Algorithm.


Choose the Dataset:
1:      Small Dataset #18
2:      Large Dataset #10
3:      Diabetes Dataset

Enter your choice (1 or 2 or 3): 3

Choose the feature selection method:
1:      Forward Selection
2:      Backward Elimination
Enter your choice (1 or 2): 1

Forward Selection chosen.

This dataset has 16 features (not including class attribute), with 520 instances.

Running Nearest Neighbor with no features and using LOOCV gets an accuracy of 61.5%
```



```
Finished Search! The best feature subset is [0, 8, 2, 5, 4, 9, 13, 11, 6, 1, 3, 7, 15, 14], with an accuracy of 98.8%.

Feature selection took 1.24 seconds.
```

Figure 8: Trace start & end for K-NN algorithm for diabetes data running Forward Selection.

Conclusion

This project demonstrated how feature selection techniques, when paired with a simple yet effective K-Nearest Neighbor classifier, can significantly improve model accuracy. By implementing both Forward Selection and Backward Elimination, we were able to identify optimal feature subsets across small, large, and real-world datasets. Forward Selection consistently performed faster, while Backward Elimination offered broader initial exploration. When applied to the Early Stage Diabetes dataset, the model achieved 98.8% accuracy, showcasing the power of careful feature subset selection in practical classification problems.