Justin Chiu

SID# 862286201

Email: jchiu030@ucr.edu

June 13, 2024

Project 2 for CS205 Spring 2024, with Dr. Eamonn Keogh

All code is original, except:

1. I used the ‘numpy’ library to handle numerical operations and array manipulations
2. I used the ‘scikit-learn’ library, specifically ‘KNeighborsClassifier’ for the k-nearest neighbors classifier and tools for cross-validation (‘KFold’ and ‘cross_val_score’).

Introduction

In this project, I implemented a k-nearest neighbors (k-NN) algorithm using Python to perform feature selection on datasets of varying sizes. The goal was to identify the most significant features that contribute to the classification accuracy. The methods used for feature selection were forward selection and backward elimination.

The datasets assigned to me were ‘CS205_small_Data__5.txt’ and ‘CS205_large_Data__40.txt’.

Methods

Forward Selection

In forward selection, we start with no features and iteratively add the feature that results in the highest increase in accuracy until no significant improvement is observed.

Backward Elimination

In backward elimination, we start with all features and iteratively remove the feature that results in the least decrease in accuracy until no significant reduction is observed.

Analysis of Feature Selection Algorithms on Datasets

Feature selection is a crucial step in the machine learning process. It aims to identify the most significant features for building a predictive model. It helps improve model performance, reduce overfitting, and decrease computation time. The performance of forward section and backward elimination is analyzed in terms of accuracy with different feature subsets.

Forward Selection on Small Dataset

The first dataset analyzed has 12 features (excluding the class attribute) and 500 instances. Using all 12 features, the initial accuracy achieved by the nearest neighbor algorithm was 72.2%. Forward Selection began by evaluating individual features, identifying feature [6] as the best initial choice with an accuracy of 84.0%

When feature [5] was added, a significant leap was observed, leading to an accuracy of 94.4%. However, adding the feature [9] to the subset [6, 5] decreased the accuracy to 91.8%.

Ultimately, the optimal feature subset was *[6, 5]*, achieving an accuracy of *94.4%*. This demonstrates that Forward Selection effectively identified a compact set of features that substantially enhanced model performance shown in Figure 1.

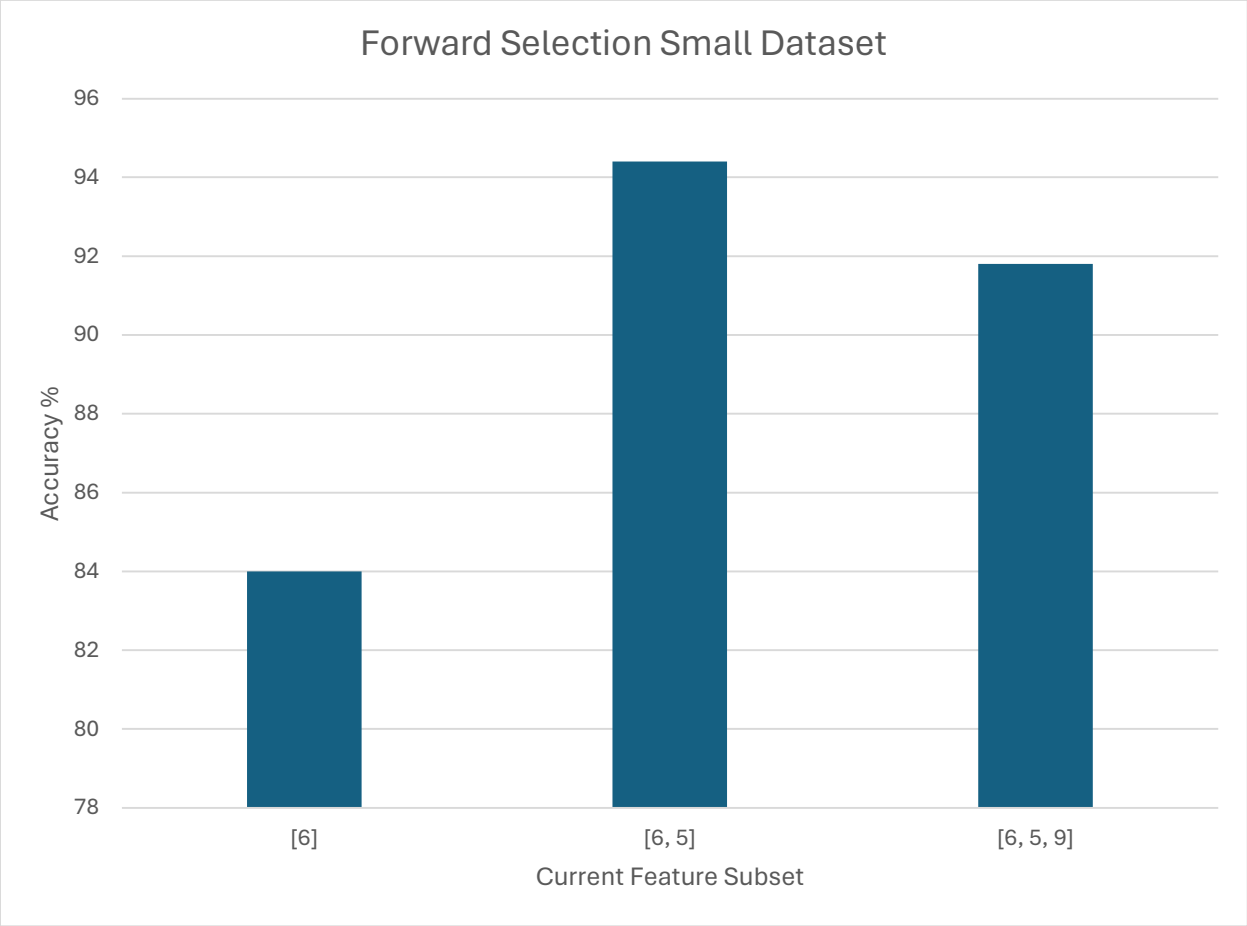

Figure 1: Accuracy for Forward Selection using Small Dataset

Backward Elimination on Small Dataset

The same small dataset was subjected to Backward Elimination, which starts with all features and iteratively removes the least significant ones. Initially, the algorithm achieved an accuracy of 72.2% with all 12 features.

The feature set [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11] has an accuracy of 76.6% when removing feature [8].

The removal of the feature [10] improved the accuracy to 77.6%. Further removals indicated the following improvements:

- Removing feature [0] improved the accuracy to 78.2%.
- Removing feature [2] improved the accuracy to 78.8%.
- Removing feature [3] improved the accuracy to 81.6%.
- Removing feature [11] improved the accuracy to 83.6%
- Removing feature [9] improved the accuracy to 85.6%.
- Removing feature [7] improved the accuracy to 87.6%.
- Removing feature [1] improved the accuracy to 90.4%.
- Removing feature [4] improved the accuracy to 94.4%.

Removing [5] decreases the accuracy significantly to 84.0%. Therefore, the optimal subset identified was *[5, 6]*, yielding an accuracy of *94.4%*.

Although the final accuracy was the same as that achieved with Forward Selection, Backward Elimination demonstrated its capability to refine the feature set by discarding irrelevant features, as shown in Figure 2.

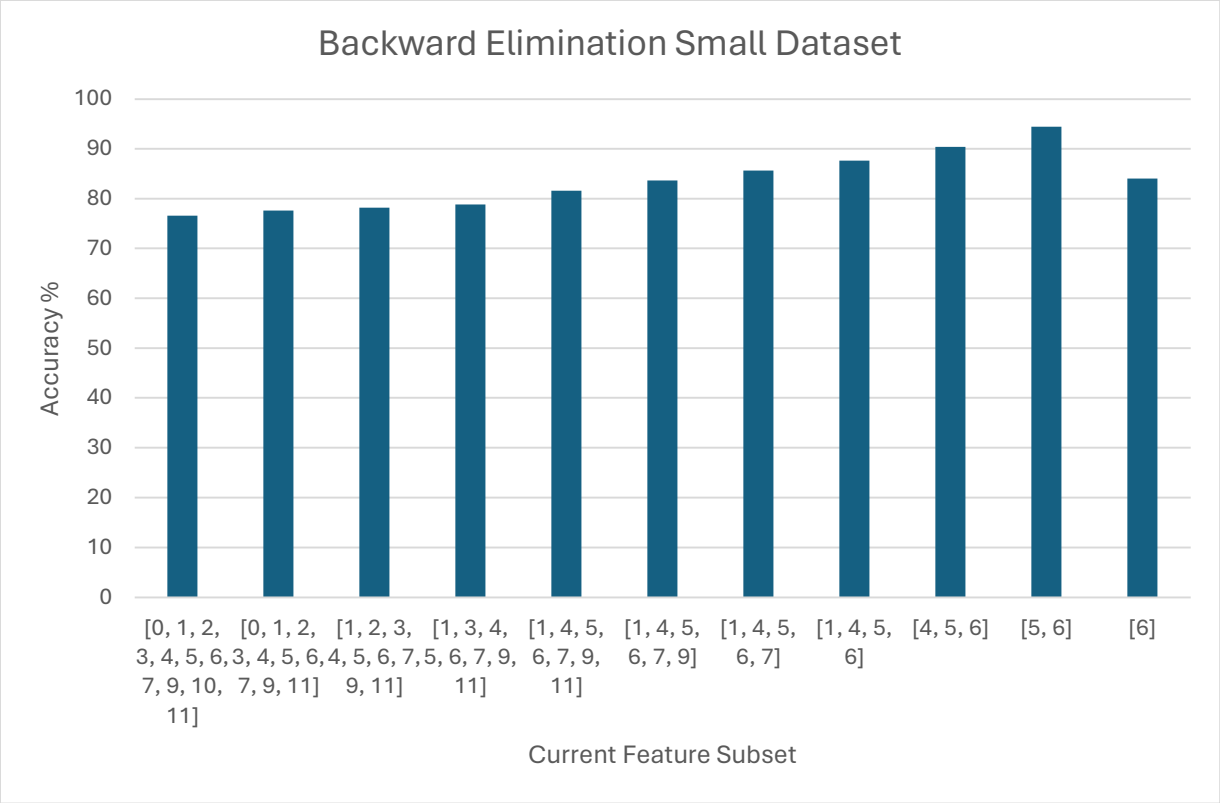

Figure 2: Accuracy for Backward Elimination using Small Dataset

Forward Selection on Large Dataset

The second dataset analyzed contains 50 features and 5000 instances. Initially, using all 50 features, the accuracy was 69.0%. Forward Selection began by identifying feature [34] as the best single feature, with an accuracy of 84.3%.

Subsequent iterations combined feature [34] with others:

- The combination of [34, 33] achieved an astounding accuracy of 97.5%.
- Further addition of feature [3] to form the set [34, 33, 3] decreased accuracy to 96.6%.

Finally, the optimal feature subset was *[34, 33]*, with an accuracy of *97.5%*. This substantial improvement illustrates the effectiveness of Forward Selection in identifying a highly relevant feature subset from a large pool of features shown in Figure 3.

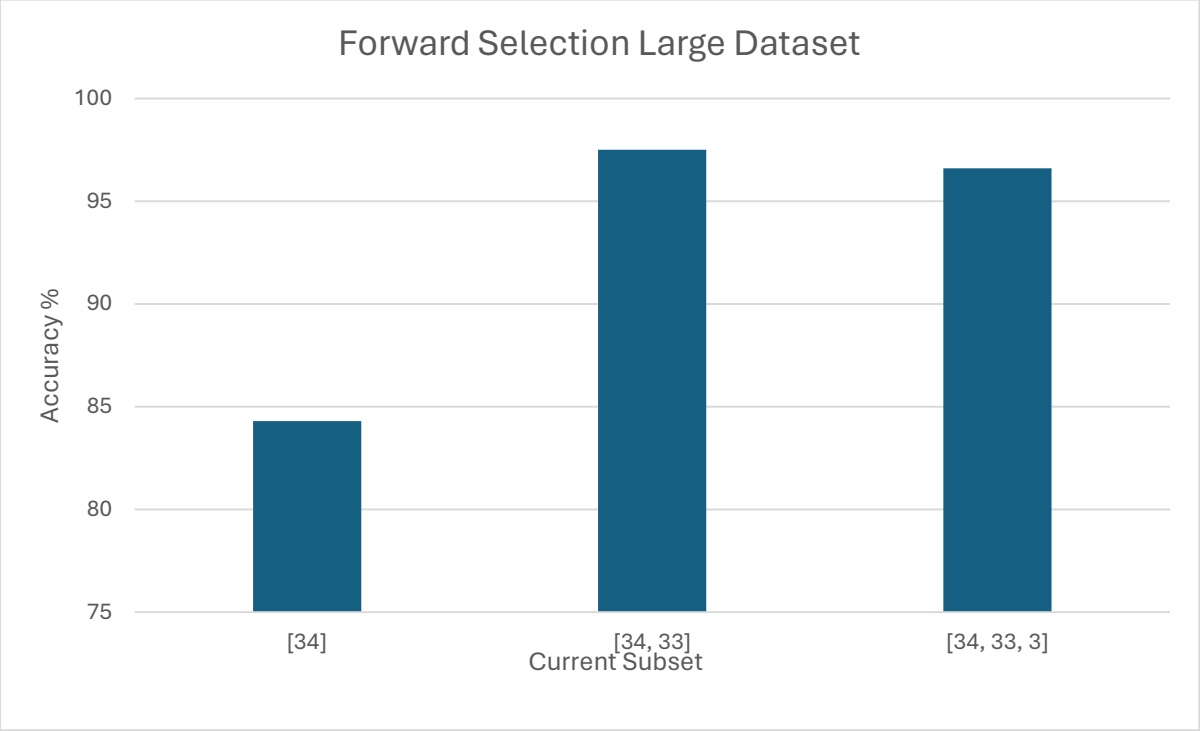

Figure 3: Accuracy for Forward Selection using Large Dataset

Backward Elimination on Large Dataset

The large dataset was also analyzed using Backward Elimination. Starting with all 50 features, the initial accuracy was 69.0%. The first elimination feature [3] increased the accuracy to 70.0%. Subsequent removals resulted in marginal accuracy changes:

- Removing feature [27] resulted in an accuracy of 70.6%.
- Removing feature [13] resulted in an accuracy of 71.1%.
- Removing feature [48] resulted in an accuracy of 71.2%.
- Removing feature [33] resulted in an accuracy of 71.4%.
- Removing feature [21] resulted in an accuracy of 71.7%.
- However removing feature [5] decreased the accuracy to 71.5%.

The final best accuracy is *71.7%* with a subset of *[0, 1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 22, 23, 24, 25, 26, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 49]*.

Despite slight improvements, Backward Elimination did not achieve the significant accuracy gains seen with Forward Selection as shown in Figure 4.

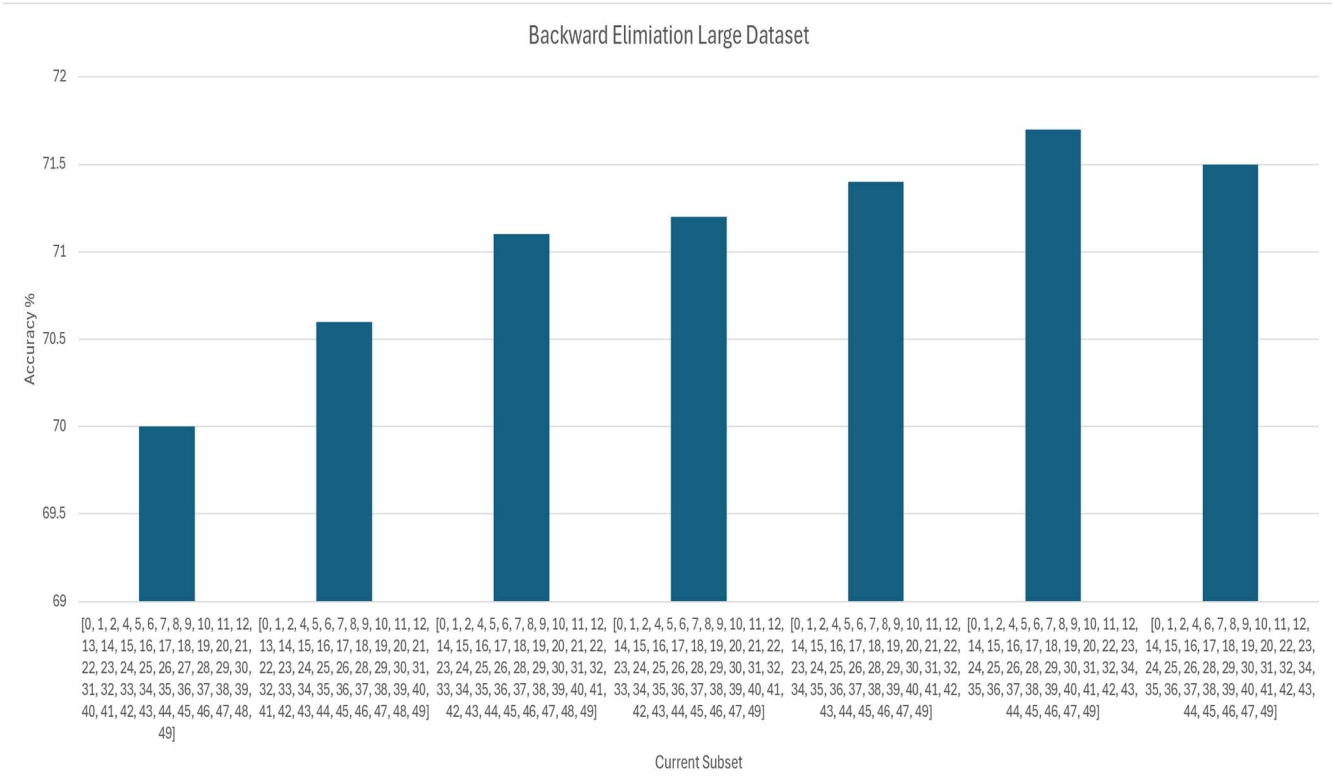

Figure 4: Accuracy for Backward Elimination using Large Dataset

Summary of Results

- Forward Selection is effective in both small and large datasets, achieving 94.4% with the small dataset and 97.5% with the large dataset.
- Backward Elimination also performs well, but with lower accuracies compared to Forward Selection when it comes to the large dataset, achieving 94.4% with the small dataset and 71.7% with the large dataset.
- Forward Selection achieved higher accuracies with more features compared to Backward Elimination, suggesting it might be more efficient for these datasets.

	Small dataset (12 features, 500 instances)	Large dataset (50 features, 5000 instances)
Forward Selection	2.26 seconds	56.45 seconds
Backward Elimination	5.70 seconds	29.64 seconds

The table above shows the execution times for two feature selection methods, Forward Selection and Backward Elimination, when applied to datasets of different sizes.

For the small dataset, the comparable execution times for Forward Selection and Backward Elimination indicate that both methods are efficient and feasible for small datasets. The longer time for Backward Elimination suggests a marginal increase in computational effort, but both methods can be executed swiftly.

The results for the large dataset reveal a stark contrast in the efficiency of the two methods. Backward Elimination demonstrated better performance. The significant increase in time for Forward Selection can be attributed to the sequential addition of features, which requires evaluating multiple combinations. In contrast, Backward Elimination starts with all features and systematically removes them, which, in this case, proved to be more efficient.

Output

Here is my output only showing the Forward Selection on the small dataset...

```
Welcome to Justin Chiu's Feature Selection Algorithm


Type in the name of the file to test: CS205_small_Data__5.txt

Type the number of the algorithm you want to run.


1) Forward Selection

2) Backward Elimination


1


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

Running nearest neighbor with all 12 features, using Forward Selection
evaluation, I get an accuracy of 72.2%.


Using feature(s) [0] accuracy is 63.4%

Using feature(s) [1] accuracy is 68.6%

Using feature(s) [2] accuracy is 62.8%

Using feature(s) [3] accuracy is 66.4%

Using feature(s) [4] accuracy is 61.6%

Using feature(s) [5] accuracy is 72.0%

Using feature(s) [6] accuracy is 84.0%

Using feature(s) [7] accuracy is 61.4%

Using feature(s) [8] accuracy is 61.2%

Using feature(s) [9] accuracy is 61.0%

Using feature(s) [10] accuracy is 67.2%

Using feature(s) [11] accuracy is 62.0%
```

Feature set [6] was best, accuracy is 84.0%

Using feature(s) [6, 0] accuracy is 81.2%

Using feature(s) [6, 1] accuracy is 80.2%

Using feature(s) [6, 2] accuracy is 80.0%

Using feature(s) [6, 3] accuracy is 80.4%

Using feature(s) [6, 4] accuracy is 80.4%

Using feature(s) [6, 5] accuracy is 94.4%

Using feature(s) [6, 7] accuracy is 79.4%

Using feature(s) [6, 8] accuracy is 76.0%

Using feature(s) [6, 9] accuracy is 78.8%

Using feature(s) [6, 10] accuracy is 77.4%

Using feature(s) [6, 11] accuracy is 81.6%

Feature set [6, 5] was best, accuracy is 94.4%

Using feature(s) [6, 5, 0] accuracy is 89.6%

Using feature(s) [6, 5, 1] accuracy is 88.8%

Using feature(s) [6, 5, 2] accuracy is 91.6%

Using feature(s) [6, 5, 3] accuracy is 91.2%

Using feature(s) [6, 5, 4] accuracy is 90.4%

Using feature(s) [6, 5, 7] accuracy is 90.8%

Using feature(s) [6, 5, 8] accuracy is 90.8%

Using feature(s) [6, 5, 9] accuracy is 91.8%

Using feature(s) [6, 5, 10] accuracy is 89.2%

Using feature(s) [6, 5, 11] accuracy is 90.8%

Elapsed time: 2.26 seconds

Finished search!! The best feature subset is [6, 5], which has an accuracy of 94.4%.

Code

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score, KFold
import time

# Load the dataset from a file
def load_dataset(filename):
    try:
        data = np.loadtxt(filename)        # load the data
    except Exception as e:
        raise ValueError(f"Error loading dataset: {e}")

    X = data[:, 1:]             # Features (all columns except the first)
    y = data[:, 0]              # Class labels (first column)
    return X, y

# Evaluation function using k-fold cross-validation
def evaluate_subset(X_subset, y, cv_splits=5):
    clf = KNeighborsClassifier(n_neighbors=1)          # 1 nearest neighbor
    kf = KFold(n_splits=cv_splits)                     # 5-fold
    scores = cross_val_score(clf, X_subset, y, cv=kf)      # 5-fold cross-
validation
    return scores.mean()            # return the mean accuracy

# Forward selection method
def forward_selection(X, y, cv_splits=5):
    start_time = time.time()                    # start time
    num_features = X.shape[1]                   # number of features
    best_subset = []                            # best feature subset
    best_accuracy = 0.0                         # best accuracy

    # Loop through all features
    while len(best_subset) < num_features:
        candidate_accuracy = []                  # list of candidate accuracies
        candidate_features = []                  # list of candidate feature sets

        # Generate the list of indices not in best_subset
        remaining_features = []
        for i in range(num_features):
            if i not in best_subset:
                remaining_features.append(i)
```

```
        # Evaluate each subset directly in the loop
        for i in remaining_features:
            subset = best_subset + [i]             # add a new feature to the
current best subset
            accuracy = evaluate_subset(X[:, subset], y, cv_splits)
            candidate_accuracy.append(accuracy)
            candidate_features.append(subset)
            print(f"Using feature(s) {subset} accuracy is {accuracy:.1%}")

        # Find the best candidate
        max_accuracy = max(candidate_accuracy)
        max_index = candidate_accuracy.index(max_accuracy)
        best_candidate = candidate_features[max_index]

        # Update the best subset and accuracy
        if max_accuracy > best_accuracy:
            best_accuracy = max_accuracy
            best_subset = best_candidate
            print(f"Feature set {best_subset} was best, accuracy is
{best_accuracy:.1%}\n")
        else:
            break

    end_time = time.time()
    print(f"Elapsed time: {end_time - start_time:.2f} seconds\n")

    return best_subset, best_accuracy

# Backward elimination method
def backward_elimination(X, y, cv_splits=5):
    start_time = time.time()                              # start time
    num_features = X.shape[1]                             # number of features
    best_subset = list(range(num_features))               # best feature subset
    best_accuracy = evaluate_subset(X, y, cv_splits)      # best accuracy

    # Loop through all features
    while len(best_subset) > 0:
        candidate_accuracy = []                           # list of candidate
accuracies
        candidate_features = []                           # list of candidate
feature sets

        # Generate the list of remaining features
        remaining_features = best_subset.copy()
```

```
        # Evaluate each subset directly in the loop
        for i in remaining_features:
            subset = remaining_features.copy()              # copy the current
best subset
            subset.remove(i)                                # remove a feature
from the current best subset
            accuracy = evaluate_subset(X[:, subset], y, cv_splits)
            candidate_accuracy.append(accuracy)
            candidate_features.append(subset)
            print(f"Using feature(s) {subset} accuracy is {accuracy:.1%}")

        # Find the best candidate
        max_accuracy = max(candidate_accuracy)
        max_index = candidate_accuracy.index(max_accuracy)
        best_candidate = candidate_features[max_index]

        # Update the best subset and accuracy
        if max_accuracy > best_accuracy:
            print(f"Feature set {best_candidate} was best, accuracy is
{max_accuracy:.1%}\n")
            best_accuracy = max_accuracy
            best_subset = best_candidate
        else:
            break

    end_time = time.time()
    print(f"Elapsed time: {end_time - start_time:.2f} seconds\n")

    return best_subset, best_accuracy

# Main function
if __name__ == "__main__":
    print("Welcome to Justin Chiu's Feature Selection Algorithm\n")
    filename = input("Type in the name of the file to test: ")
    X, y = load_dataset(filename)

    algorithms = {
        "1": "Forward Selection",
        "2": "Backward Elimination"
    }

    selection = input("Type the number of the algorithm you want to run. \n\n" +
                      "1) Forward Selection\n" +
                      "2) Backward Elimination\n\n")
```

```
    if selection not in algorithms:
        raise ValueError("Invalid algorithm selection.")

    print(f"\nThis dataset has {X.shape[1]} features (not including the class
attribute), with {X.shape[0]} instances.")

    accuracy_all_features = evaluate_subset(X, y)
    print(f"Running nearest neighbor with all {X.shape[1]} features, using
{algorithms[selection]} evaluation, I get an accuracy of
{accuracy_all_features:.1%}.\n")

    if selection == "1":
        best_subset, best_accuracy = forward_selection(X, y)
    elif selection == "2":
        best_subset, best_accuracy = backward_elimination(X, y)

    print(f"\nFinished search!! The best feature subset is {best_subset}, which
has an accuracy of {best_accuracy:.1%}.\n")
```