Project 2: Feature Selection with Nearest Neighbor

Student Name1: Thien Pham SID: 862107055 Lecture Session: 2

Solution: <the datasets are your uniquely assigned datasets>

Dataset	Best Feature Set	Accuracy
Small Number: <insert your small dataset number>	Forward Selection = {1, 5, 2}	0.97
	Backward Elimination = {1, 5, 9}	0.94
	Custom Algorithm = Not implemented	NA
Large Number: 90	Forward Selection = {32, 26}	0.96
	Backward Elimination = {1, 17, 32}	0.85
	Custom Algorithm = Not implemented	NA

--------------------------------<Begin Report>---------------------------------- In completing this project, I consulted following resources: The train/split/test code from the TA

Contribution of each student in the group: Not Applicable

I. Introduction

The project is about the implementation of the Nearest Neighbor classifier using greedy feature search with leave-one-out validation for accuracy calculation. The objective is to identify the strengths and weaknesses of greedy forward selection and backward selection.

II. Challenges

In this project, I struggled with writing the feature selection algorithm. I understood the concept of selecting the highest accuracy at each level in a tree, but implementing it took me a bit of time to process. I ended up setting up multiple lists that may have been redundant. The multiple lists were a result of trying to find a roundabout way from dropping a row in a dataframe. The iterative loop would no longer work since a row was deleted. I was also fairly new to python so it was difficult to get my program to run at times. Another major challenge I faced was creating the feature selection and the classifier along with the validation method in parts. It was my first time creating a feature selection, classifier, and validation of any sort, so it was difficult to visualize how to structure everything. When the accuracy of a set and a simpler set was found, I was unable to choose the simpler set on the backward selection, it would just keep the first set that was found with that accuracy. I also had a tough time visualizing how data would be read from the csv file. Basically, everything felt unpredictable, and I had to go back and revise my feature selection algorithm since I thought that accuracy of combination was calculated as the sum of the accuracies of the individual features divided by the total number of features. After finishing the classifier class and the validation class, I realized that the combination accuracies were calculated by training the data with the selected features, and had to adjust my feature selection algorithms.

III. Code Design

```
import pandas as pd

import numpy as np




def greedySearchForwardSelection(numFeatures):

    df = pd.read_csv(file_name,sep=r'\s+',header=None) # setup
correct df

    print(f"This dataset has {df.shape[1]-1} features (not including
```

```
the class attribute), with {df.shape[0]} instances.")

    candidateFeatures = pd.DataFrame(columns=['Accuracy',
'Feature']) # load df for all feature's individual accuracies

    NearestNeighbor = NNClassifier()

    LOOV_validation = LOOV()



    # load in all individual accuracies and corresponding feature

    for i in range(numFeatures):

        featureEvaluation = LOOV_validation.validate([i+1],
NearestNeighbor, df);

        candidateFeatures.loc[i, 'Accuracy'] = featureEvaluation;

        candidateFeatures.loc[i, 'Feature'] = i+1;


    # empty set for selected features, use to keep track of optimal
feature combination

    selectedFeatures = pd.DataFrame(columns=['Accuracy', 'Feature'])

    # keeps track of current best overall feature combination
accuracy

    bestSelectedFeatures = []

    labels = df.iloc[:,0]

    two_counts = 0

    one_counts = 0

    # count most common class

    for i in range(len(labels)):

        if (labels.iloc[i] == 1):

            one_counts += 1
```

```
        if (labels.iloc[i] == 2):

            two_counts += 1

    if (two_counts > one_counts):

        defaultRate = two_counts/len(df)

    else:

        defaultRate = one_counts/len(df)

    print(f"Please wait while I normalize the data... Done!")

    print(f"Running nearest neighbor with no features (default
rate), using “leaving-one-out” evaluation, I get an accuracy of
{defaultRate * 100:.1f}%")


    currAccuracy = 0

    bestAccuracy = defaultRate

    bestAccuracyThisIteration = defaultRate # tracker for local
optima

    print(f"Beginning search.")

    while len(selectedFeatures) < numFeatures:

        bestFeature = -1

        localBestAccuracy = -1

        localBestFeatures = []

        for i in range(len(candidateFeatures)):

            consideredFeatures =
selectedFeatures['Feature'].tolist()  # Hold best combination from
previous iteration

            consideredFeatures.append(candidateFeatures.loc[i,
'Feature'])  # Append the current feature being tested with best
combination from previous iteration
```

```
            currAccuracy =
LOOV_validation.validate(consideredFeatures, NearestNeighbor, df)

            # print features currently being tested

            print(f"    Using feature(s) {{{', '.join(map(str,
consideredFeatures))}}} accuracy is {currAccuracy * 100:.1f}%")

            if localBestAccuracy < currAccuracy:

                localBestAccuracy = currAccuracy # Select locally
best accuracy

                localBestFeatures = consideredFeatures

                bestFeature = i


            if bestAccuracy < localBestAccuracy:

                bestAccuracy = localBestAccuracy

                bestSelectedFeatures = localBestFeatures




        if bestFeature != -1:

            selectedFeatures = pd.concat([selectedFeatures,
candidateFeatures.iloc[[bestFeature]].reset_index(drop=True)],
ignore_index=True)

            candidateFeatures =
candidateFeatures.drop(bestFeature).reset_index(drop=True)

            oldBest = bestAccuracyThisIteration # hold previous best
for comparison to indicate accuracy drop

            bestAccuracyThisIteration =
LOOV_validation.validate(localBestFeatures, NearestNeighbor, df)

            if (oldBest > bestAccuracyThisIteration):

                print("(Warning, Accuracy has decreased! Continuing
```

```
search in case of local maxima)")
            print(f"Feature set {{{', '.join(map(str,
localBestFeatures))}}} was best, accuracy is
{(bestAccuracyThisIteration)* 100:.1f}%")



        else:
            break # Stop when no more features left


    print(f"Finished search!! The best feature subset is {{{',
'.join(map(str, bestSelectedFeatures))}}}, which has an accuracy of
{bestAccuracy* 100:.1f}%")


def greedySearchBackwardSelection(numFeatures):
    df = pd.read_csv(file_name,sep=r'\s+',header=None) # setup
correct df
    print(f"This dataset has {df.shape[1]-1} features (not including
the class attribute), with {df.shape[0]} instances.")
    candidateFeatures = pd.DataFrame(columns=['Accuracy',
'Feature']) # load df for all feature's individual accuracies
    NearestNeighbor = NNClassifier()
    LOOV_validation = LOOV()


    # load in all individual accuracies and corresponding feature
    for i in range(numFeatures):
        featureEvaluation = LOOV_validation.validate([i+1],
NearestNeighbor, df);
        candidateFeatures.loc[i, 'Accuracy'] = featureEvaluation;
```

```
        candidateFeatures.loc[i, 'Feature'] = i+1;


    selectedFeatures = pd.DataFrame(columns=['Accuracy', 'Feature'])

    bestSelectedFeatures = []

    labels = df.iloc[:,0]

    two_counts = 0

    one_counts = 0

    for i in range(len(labels)):

        if (labels.iloc[i] == 1):

            one_counts += 1

        if (labels.iloc[i] == 2):

            two_counts += 1

    if (two_counts > one_counts):

        defaultRate = two_counts/len(df)

    else:

        defaultRate = one_counts/len(df)


    print(f"Please wait while I normalize the data... Done!")

    print(f"Running nearest neighbor with no features (default
rate), using “leaving-one-out” evaluation, I get an accuracy of
{defaultRate * 100:.1f}%")

    currAccuracy = 0

    bestAccuracy = defaultRate

    bestAccuracyThisIteration = defaultRate

    print(f"Beginning search.")
```

```
    while len(selectedFeatures) < numFeatures:
        bestFeature = -1
        localBestAccuracy = -1


        for i in range(len(candidateFeatures)):
            consideredFeatures =
candidateFeatures['Feature'].tolist()  # Hold best combination from
previous iteration
            consideredFeatures.remove(candidateFeatures.loc[i,
'Feature'])  # Append the current feature being tested with best
combination from previous iteration
            currAccuracy = currAccuracy =
LOOV_validation.validate(consideredFeatures, NearestNeighbor, df)
            # print features currently being tested
            print(f"    Using feature(s) {{{', '.join(map(str,
consideredFeatures))}}} accuracy is {currAccuracy * 100:.1f}%")


            if localBestAccuracy < currAccuracy:
                localBestAccuracy = currAccuracy # Select locally
best accuracy
                localBestFeatures = consideredFeatures
                bestFeature = i


            if bestAccuracy < localBestAccuracy:
                bestAccuracy = localBestAccuracy
                bestSelectedFeatures = localBestFeatures
```

```
        if bestFeature != -1:
            selectedFeatures = pd.concat([selectedFeatures,
candidateFeatures.iloc[[bestFeature]].reset_index(drop=True)],
ignore_index=True)
            candidateFeatures =
candidateFeatures.drop(bestFeature).reset_index(drop=True)
            oldBest = bestAccuracyThisIteration # hold previous best
for comparison to indicate accuracy drop
            bestAccuracyThisIteration =
LOOV_validation.validate(localBestFeatures, NearestNeighbor, df)
            if (oldBest > bestAccuracyThisIteration):
                print("(Warning, Accuracy has decreased! Continuing
search in case of local maxima)")
            print(f"Feature set {{{', '.join(map(str,
localBestFeatures))}}} was best, accuracy is
{bestAccuracyThisIteration* 100:.1f}%")

        else:
            break # Stop when no more features left
    print(f"Finished search!! The best feature subset is {{{',
'.join(map(str, bestSelectedFeatures))}}}, which has an accuracy of
{bestAccuracy* 100:.1f}%")



class NNClassifier():
    def __init__(self):
```

```
        self.X_norm = None

        self.means = None

        self.std = None

        self.train_x = None

        self.train_y = None


    def train(self, X):

        self.train_x = X.iloc[:,1:]

        self.means = self.train_x.mean()

        self.std = self.train_x.std()

        self.X_norm = (self.train_x - self.means)/self.std

        self.train_y = X.iloc[:,0] # store labels


    def test(self, X):

        # Normalized instance

        train_x = X.iloc[1:]

        X_norm  = (train_x - self.means)/self.std

        X_norm = X_norm.values.reshape(1,-1)

        distances = np.sqrt(np.sum((self.X_norm-X_norm)**2, axis = 
1))      #Euclidean distance calculation axis needs to be 1 for 
correct element operations


        predicted_label_index = np.argmin(distances) 
#Getting the index of the nearest neighbor for euclidean distance


        predicted_val = self.train_y.iloc[predicted_label_index]
```

```
#Getting Predicted value from the nn index
        return predicted_val


class LOOV():
    #leave one instance out, train_x is set to the rest
    def validate(self, feature, classifier, dataset):
        # count correct predictions
        correct = 0
        # include labels in selected feature for classifier to store
        selectedFeatures = [0] + feature
        datasetWithOnlyLabelAndSelectedFeature = dataset.iloc[:,
selectedFeatures]
        y_actual = dataset.iloc[:,0]
        for i in range(len(dataset)):
            leaveOutIndex = i

classifier.train(datasetWithOnlyLabelAndSelectedFeature.drop(leaveOu
tIndex, axis = 0)) # leave out row i
            y_pred =
classifier.test(datasetWithOnlyLabelAndSelectedFeature.iloc[leaveOut
Index])
            if (y_pred == y_actual.iloc[leaveOutIndex]):
                correct += 1
        Accuracy = correct/len(dataset)
        return Accuracy


print(f"Welcome to Thien Pham Feature Selection Algorithm.")
```

```python
print(f"Type in the name of the file to test : ")
file_name = input()
print(f"Type the number of the algorithm you want to run.\n")
print(f"    Forward Selection")
print(f"    Backward Selection")
algorithm_type = int(input())
df = pd.read_csv(file_name,sep=r'\s+',header=None)
if (algorithm_type == 1):
    greedySearchForwardSelection(df.shape[1]-1)
if (algorithm_type == 2):
    greedySearchBackwardSelection(df.shape[1]-1)
```

IV. Dataset details

The General Small Dataset: 10 features, 100 instances

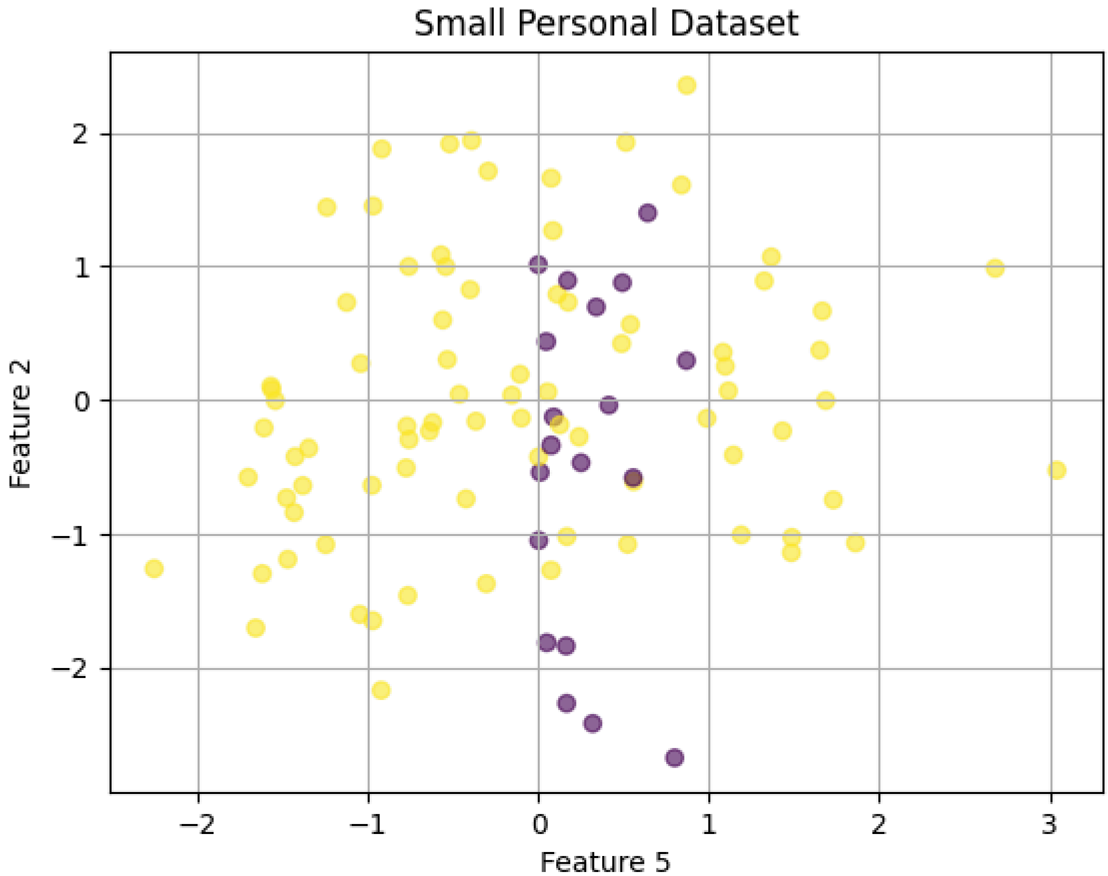
The General Large Dataset: 40 features, 1000 instances

Your Personal Small Dataset: 10 features, 100 instances

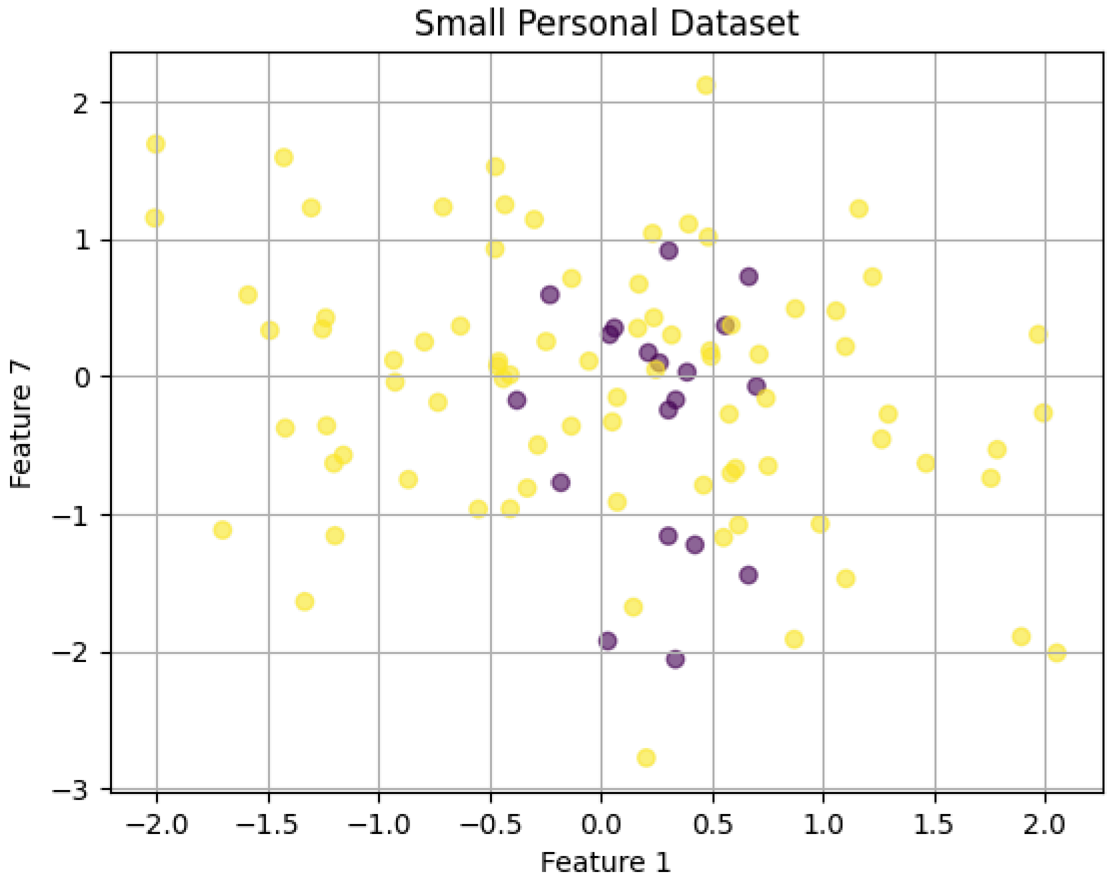
Your Personal Large Dataset: 40 features, 1000 instances

Plot some features and color code them by class and explore your dataset.

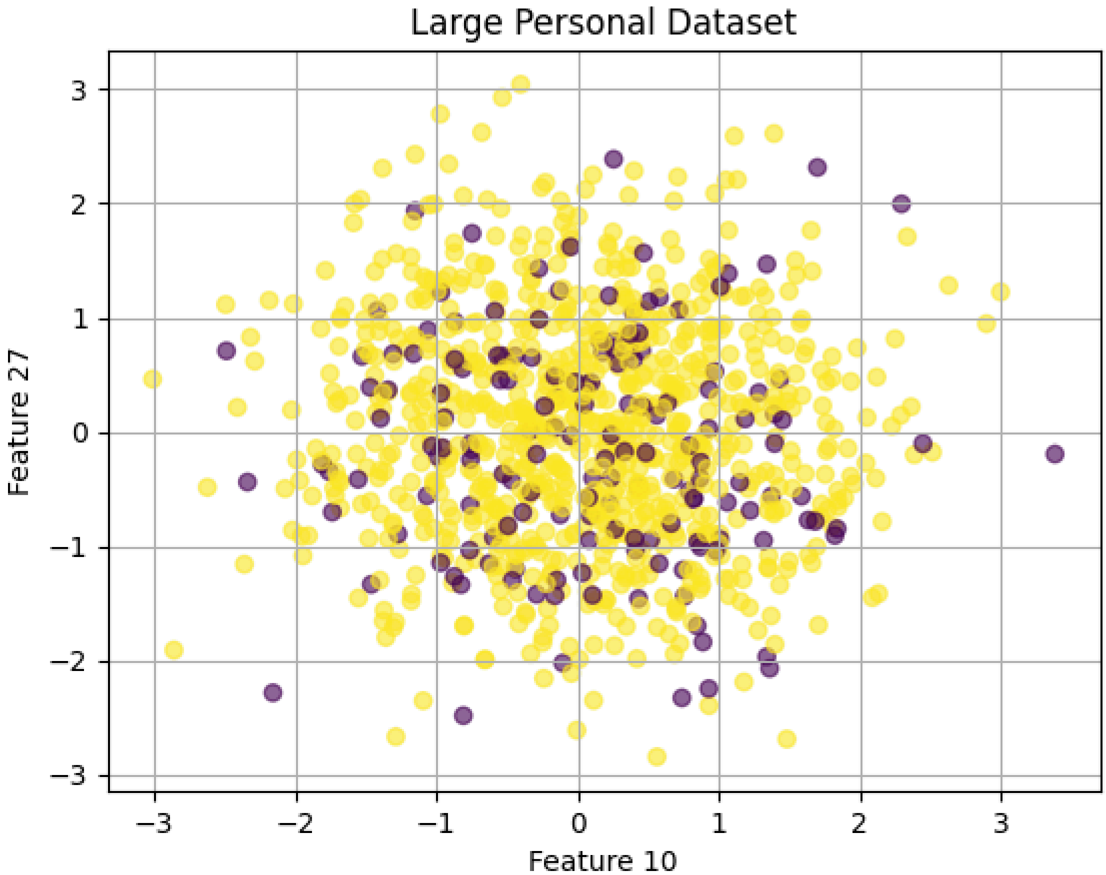
Feature 5 vs Feature 2(Small):



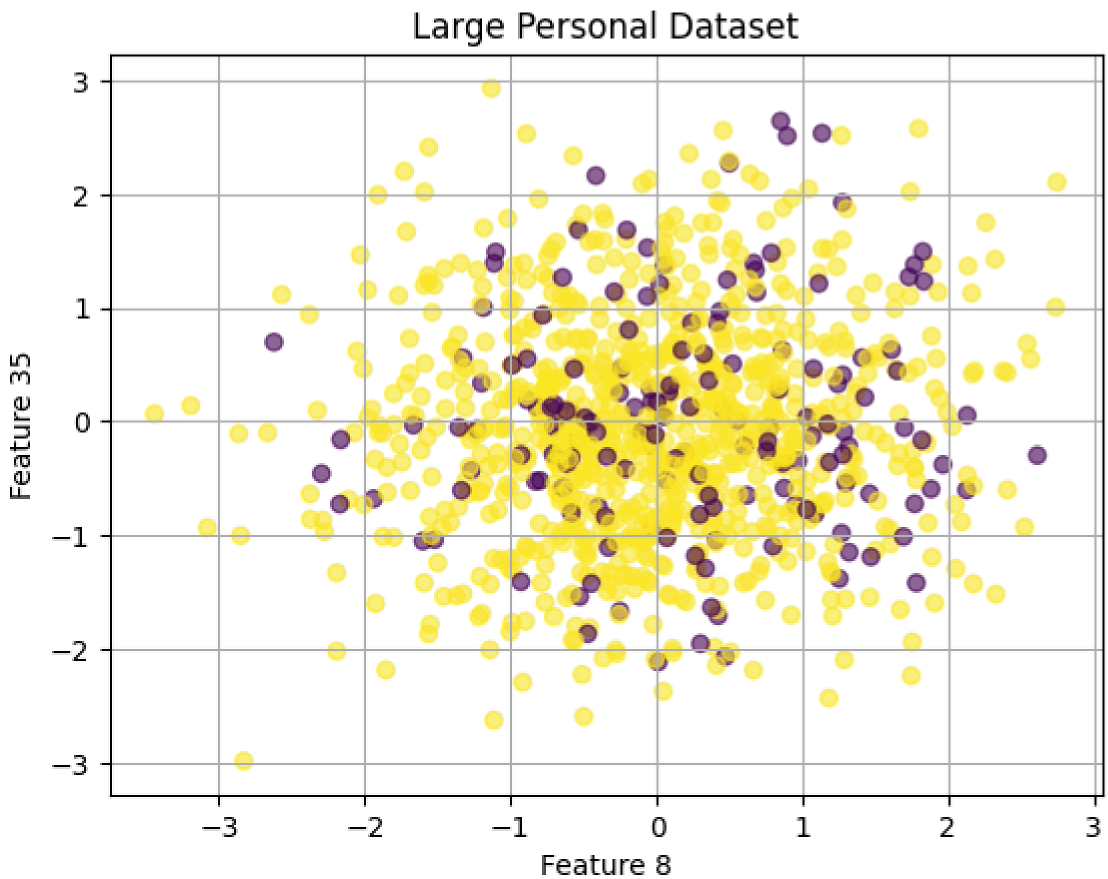
Feature 1 vs Feature 7(Small):



Feature 10 vs Feature 27(Large):



Feature 8 vs Feature 35:



V. Algorithms

1. Forward Selection
    - Create three sets: one with all features, and one empty set that stores local optima, and one that stores global optima. Store the combination accuracies at each specific level–the n number of features used, for example level 2 means a maximum of 2 features are used–in the local optima set. Keep track of the global optima features and accuracies using the global optima set. Run the algorithm until the deepest level is reached.
2. Backward Elimination
    - Create three sets: one with all features, a copy of the full set that stores local optima, and one that stores global optima. Store the combination accuracies at each specific level–the n number of features used, for example level 2 means a maximum of 2 features are used–in the local optima set. Keep track of the global optima features and accuracies using the global optima set. Run the algorithm until the deepest level is reached.

VI. Analysis

The accuracy of no feature selection was consistently lower than accuracy with feature selection. The accuracy for the forward selection was consistently higher than backward elimination. The forward selection is better at selecting the best features. It is much more focused on the strength of individual features, since it must choose the best combination starting from an empty set. The backward selection may remove strong individual features near the start of the algorithm, resulting in a lower accuracy set by the end of the algorithm. However, the two algorithms should be executed together when searching for features as they likely converge onto a feature set that contains multiple similar features. The strongest features can be found this way.

VII. Conclusion

The forward selection algorithm produced better results. This is a result of the non-optimality of greedy algorithms. However, both algorithms should be ran until they converge to find the most defining features, as both algorithms should produce sets with high similarity. In the future, I could improve my runtime by using a priority queue to keep track of the local and global optima's features and accuracy as a tuple.

VIII. Trace of your small dataset

```
Welcome to Thien Pham Feature Selection Algorithm.

Type in the name of the file to test :

Type the number of the algorithm you want to run.


    Forward Selection

    Backward Selection

This dataset has 10 features (not including the class attribute),
with 100 instances.

Please wait while I normalize the data... Done!

Running nearest neighbor with no features (default rate), using
"leaving-one-out" evaluation, I get an accuracy of 81.0%

Beginning search.

    Using feature(s) {1} accuracy is 83.0%

    Using feature(s) {2} accuracy is 76.0%
```

```
    Using feature(s) {3} accuracy is 66.0%
    Using feature(s) {4} accuracy is 69.0%
    Using feature(s) {5} accuracy is 77.0%
    Using feature(s) {6} accuracy is 73.0%
    Using feature(s) {7} accuracy is 62.0%
    Using feature(s) {8} accuracy is 69.0%
    Using feature(s) {9} accuracy is 62.0%
    Using feature(s) {10} accuracy is 61.0%
Feature set {1} was best, accuracy is 83.0%
    Using feature(s) {1, 2} accuracy is 73.0%
    Using feature(s) {1, 3} accuracy is 75.0%
    Using feature(s) {1, 4} accuracy is 65.0%
    Using feature(s) {1, 5} accuracy is 94.0%
    Using feature(s) {1, 6} accuracy is 81.0%
    Using feature(s) {1, 7} accuracy is 79.0%
    Using feature(s) {1, 8} accuracy is 75.0%
    Using feature(s) {1, 9} accuracy is 72.0%
    Using feature(s) {1, 10} accuracy is 72.0%
Feature set {1, 5} was best, accuracy is 94.0%
    Using feature(s) {1, 5, 2} accuracy is 97.0%
    Using feature(s) {1, 5, 3} accuracy is 91.0%
    Using feature(s) {1, 5, 4} accuracy is 88.0%
    Using feature(s) {1, 5, 6} accuracy is 91.0%
    Using feature(s) {1, 5, 7} accuracy is 87.0%
    Using feature(s) {1, 5, 8} accuracy is 91.0%
    Using feature(s) {1, 5, 9} accuracy is 94.0%
    Using feature(s) {1, 5, 10} accuracy is 95.0%
Feature set {1, 5, 2} was best, accuracy is 97.0%
    Using feature(s) {1, 5, 2, 3} accuracy is 87.0%
    Using feature(s) {1, 5, 2, 4} accuracy is 87.0%
```

Using feature(s) {1, 5, 2, 6} accuracy is 94.0%

Using feature(s) {1, 5, 2, 7} accuracy is 86.0%

Using feature(s) {1, 5, 2, 8} accuracy is 86.0%

Using feature(s) {1, 5, 2, 9} accuracy is 88.0%

Using feature(s) {1, 5, 2, 10} accuracy is 88.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)

Feature set {1, 5, 2, 6} was best, accuracy is 94.0%

Using feature(s) {1, 5, 2, 6, 3} accuracy is 90.0%

Using feature(s) {1, 5, 2, 6, 4} accuracy is 87.0%

Using feature(s) {1, 5, 2, 6, 7} accuracy is 84.0%

Using feature(s) {1, 5, 2, 6, 8} accuracy is 85.0%

Using feature(s) {1, 5, 2, 6, 9} accuracy is 84.0%

Using feature(s) {1, 5, 2, 6, 10} accuracy is 85.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)

Feature set {1, 5, 2, 6, 3} was best, accuracy is 90.0%

Using feature(s) {1, 5, 2, 6, 3, 4} accuracy is 83.0%

Using feature(s) {1, 5, 2, 6, 3, 7} accuracy is 77.0%

Using feature(s) {1, 5, 2, 6, 3, 8} accuracy is 80.0%

Using feature(s) {1, 5, 2, 6, 3, 9} accuracy is 76.0%

Using feature(s) {1, 5, 2, 6, 3, 10} accuracy is 84.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)

Feature set {1, 5, 2, 6, 3, 10} was best, accuracy is 84.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 4} accuracy is 77.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 7} accuracy is 76.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 8} accuracy is 79.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 9} accuracy is 77.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)

Feature set {1, 5, 2, 6, 3, 10, 8} was best, accuracy is 79.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 8, 4} accuracy is 75.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 8, 7} accuracy is 80.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 8, 9} accuracy is 79.0%

Feature set {1, 5, 2, 6, 3, 10, 8, 7} was best, accuracy is 80.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 8, 7, 4} accuracy is 75.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 8, 7, 9} accuracy is 76.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)

Feature set {1, 5, 2, 6, 3, 10, 8, 7, 9} was best, accuracy is 76.0%

Using feature(s) {1, 5, 2, 6, 3, 10, 8, 7, 9, 4} accuracy is 71.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)

Feature set {1, 5, 2, 6, 3, 10, 8, 7, 9, 4} was best, accuracy is 71.0%

Finished search!! The best feature subset is {1, 5, 2}, which has an accuracy of 97.0%