Project 2: Feature Selection with Nearest Neighbor

Krishaan Patel	kpate135	SID: 862181397	Lecture Session: 001
Calvin Trujillo	ctruj011	SID: 862196581	Lecture Session: 001
Ethan Liu	eliu058	SID: 862168527	Lecture Session: 001
Christian Lo	clo060	SID:	Lecture Session: 001

Solution:

Dataset	Best Feature Set	Accuracy
Small Number: 2	Forward Selection = {3,9}	93%
	Backward Elimination = {10,9,7,2,1}	85%

Dataset	Best Feature Set	Accuracy
Large Number: 2	Forward Selection = {31,21}	97.6%
	Backward Elimination = {40,39,38,37,36,35,34,33,30,13,12,11,10,9,8,6,5,4,3,2,1,14,15,16,17,18,19,20,21,22,23,24,25,26,27,29}	75.9%

In completing the project, I consulted following resources:

General C++ Library Explanations: https://cplusplus.com/reference/

https://en.cppreference.com/w/cpp/links/libs

C++ STL Iterators: https://www.geeksforgeeks.org/iterators-c-stl/

Contribution of each student in the group:

Krishaan Patel:

- The part 1 implementation was designing by following the given pseudocode and altering it for a C++ implementation. The forward_selection_search and backward_elimination_search method were implemented by just taking in an integer N representing features, and then traversing the levels of the search tree and finding which addition/removal of each of the N features yielded the highest accuracy.
    - Utilized unordered_set<int> to ensure O(1) lookup time and efficient storing and erasing of features to insert/remove at each level of the search.
- The Classifier class is designed to have the Train method that stores the current set of training instances, the Test method that stores the class label and features of the instance to predict, and the load function to handle reading in from the file.
- The Validator class handles the leave_one_out_cross_validation method to take in data and a set of features to test the nearest neighbor classifier on and subsequently return a predicted accuracy.

Calvin Trujillo:

- For part 1 - 3, I was helping debug some parts of the code and check where the outputs were going wrong.
    - If a bug was found, I would attempt to fix the problem on my end and message the group when the solution was found.
- Helped plot the features in MatLab
    - Learned MatLab to plot data
- Looked through C++ Library to figure out what we would need to store the data.
    - Researched a small bit of the fstream library to be able to access the file we need and to enter the data into the 2D vector we need.

Ethan Liu:

- For parts 1-3, I helped implement and debug parts of the code when outputs were showing up wrong.
- Helped Krishaan figure out how to implement the forward_selection_search and backward_elimination_search.
- Helped Krishaan implement the classifier class where the train method stores current instances. Also helped implement the validator class where it takes in data and features to test NN as well.
- Used MatLab's to plot data as well.

I. Introduction

Machine learning is an important part of creating an AI system. The AI takes all the information being fed into it and creates its own criteria to identify the classes which an object belongs to. This allows the AI to identify certain new objects and place them into different classes. In this project, we will learn the basics of Machine Learning and train the AI to identify which feature sets are the most important to help classify an object.

II. Challenges

- One of the challenges faced was the handling of manipulation and accessing of the dataset.
    - Because of our implementation being in C++, we decided to read in our datasets and store them within a 2D-vector data-structure (vector<vector<double>> data). However, in C++ there are no standard methods for direct lookup and manipulation of certain rows/columns of a 2D vector/array. For Python, we would be able to make use of Pandas dataframes and easily access or manipulate certain rows and columns. However, in C++ we had to make use of iterators and nested loops to copy over only certain columns (to get specific features). This was more tedious and required some additional memory and runtime to store altered copies of the dataset.
- Another challenge was determining the best data structure to use across each scenario. Specifically, when it came to storing and retrieving the set of features at any given time in the algorithm, we initially worked with integers denoting the number of each feature, however needed to use an unordred_set<int> in order to have efficient lookup time and quick storage/removal.
- Another challenge was making the program run in parallel with other instances of itself. We wanted to do this to get all the sets done as quickly as possible. This run does not identify which algorithms run faster than others. (simply was to get data)
    - This was a bit of a hardware limitation on our end as some of the processing power was being used on other items.
    - Running the program got a bit faster when we ended tasks like chrome or firefox on our systems.
        - Allowed the program to run a bit more smoother and faster in parallel

III. Code Design

<code documentation>

1. Classifier Class

- The Classifier class is responsible for holding the training instances in memory and then predicting the class of an inputted test instance using Nearest Neighbor Classification.
    - **Private data member:** `vector<vector<double>> train_data;`

    - Stores the current set of training instances (class labels and features) from the dataset
- **Train:** `void Train(vector<vector<double>> train_data)`
    - The "train" method is responsible for taking in as input the set of training instances and storing them in our private *train_data* variable.
    - The classifier will use the training data to later compare to the *test_instance* and compute Euclidean Distances to find the nearest neighbor.
- **Test:** `double Test(vector<double> test_instance)`
    - The "test" method takes as input a *test_instance* and compares the *test_instance* features to each instance in the *train_data*, calculating the Euclidean Distance in order to find the nearest neighbor.
    - We return a classification of the passed in *test_instance* which will be the class of the nearest neighbor found.
- **Load:** `vector<vector<double>> load(string fileName)`
    - The "load" method takes as input a string for the file name of the dataset of interest.
    - Reads in data from the given file, using *ifstream* and *stringstream* to retrieve each line and blank space delimited "word" from the file.
        - Constructs and returns a 2D-vector/array of *doubles* to represent the dataset in separate rows and columns.
        - *Stringstream* casts each IEEE-754 formatted value to a *double* for the sake of manipulating the data.

2. Validator Class

- The Validator class is used as our evaluation module to test our Nearest Neighbor Classifier on specific feature subsets of the dataset and determine an accuracy value.
    - **leave_one_out_cross_validation:**
      `double leave_one_out_cross_validation(vector<vector<double>> data, unordered_set<int> current_set, int feature_to_add)`
        - The "leave_one_out_cross_validation" method is responsible for testing our Nearest Neighbor classifier.
            - It goes through each of the instances, taking a single instance out and using it as our test instance in the *Test* method. Assigns the rest of the instances as training data in our *Train* method.
            - We perform our Nearest Neighbor Classifier and compare the predicted class label to our test instance's actual class label.
                - We count how many correct classifications are made after repeating the process for each instance in the data.
                    - We evaluate and return an accuracy based on the amount of correct classifications divided by our total number of instances.

- **get_data_for_specific_features:**
  `vector<vector<double>> get_data_for_specific_features(vector<vector<double>> data, unordered_set<int> features)`
    - The "get_data_for_specific_features" method obtains a copy of the dataset with all instances but only including specified features
    - The data and an *unordered_set<int>* representing the list of desired features are passed as input to the function.
      - A new dataset is constructed and returned by pushing cells whose indices are corresponding to the *integers* within our *unordered_set*.of features.

3. **Feature Search Algorithms:**

- Find the best feature subset that returns the highest accuracy on our Nearest Neighbor Classifier.
  - **forward_selection_search:** `void forward_selection_search(vector<vector<double>> data)`
    - Takes as input the data, and traverses levels of the search tree greedily adding features based on the highest accuracy found.
      - Stops when accuracy decreases and returns the highest accuracy and corresponding feature subset found.
  - **backward_elimination_search:** `void backward_elimination_search(vector<vector<double>> data)`
    - Takes as input the data, and traverses levels of the search tree greedily removing features based on the highest accuracy found.
      - Stops when accuracy decreases and returns the highest accuracy and corresponding feature subset found.

IV. Dataset details

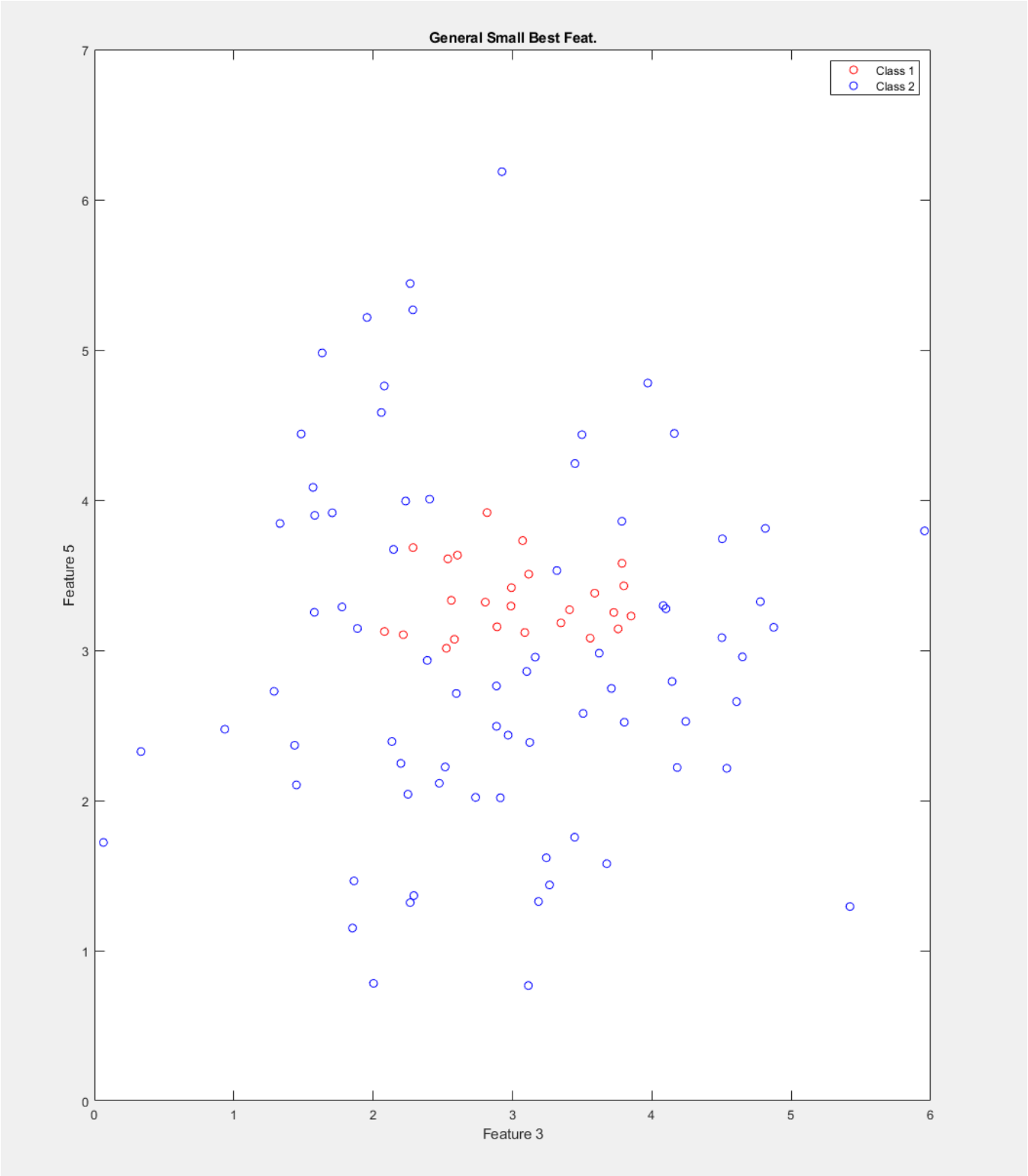
Dataset	Number of Features	Number of Instances
The General Small Dataset	10	100
The General Large Dataset	40	1000
Our Personal Small Dataset (2)	10	100
Our Personal Large Dataset (2)	40	1000

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

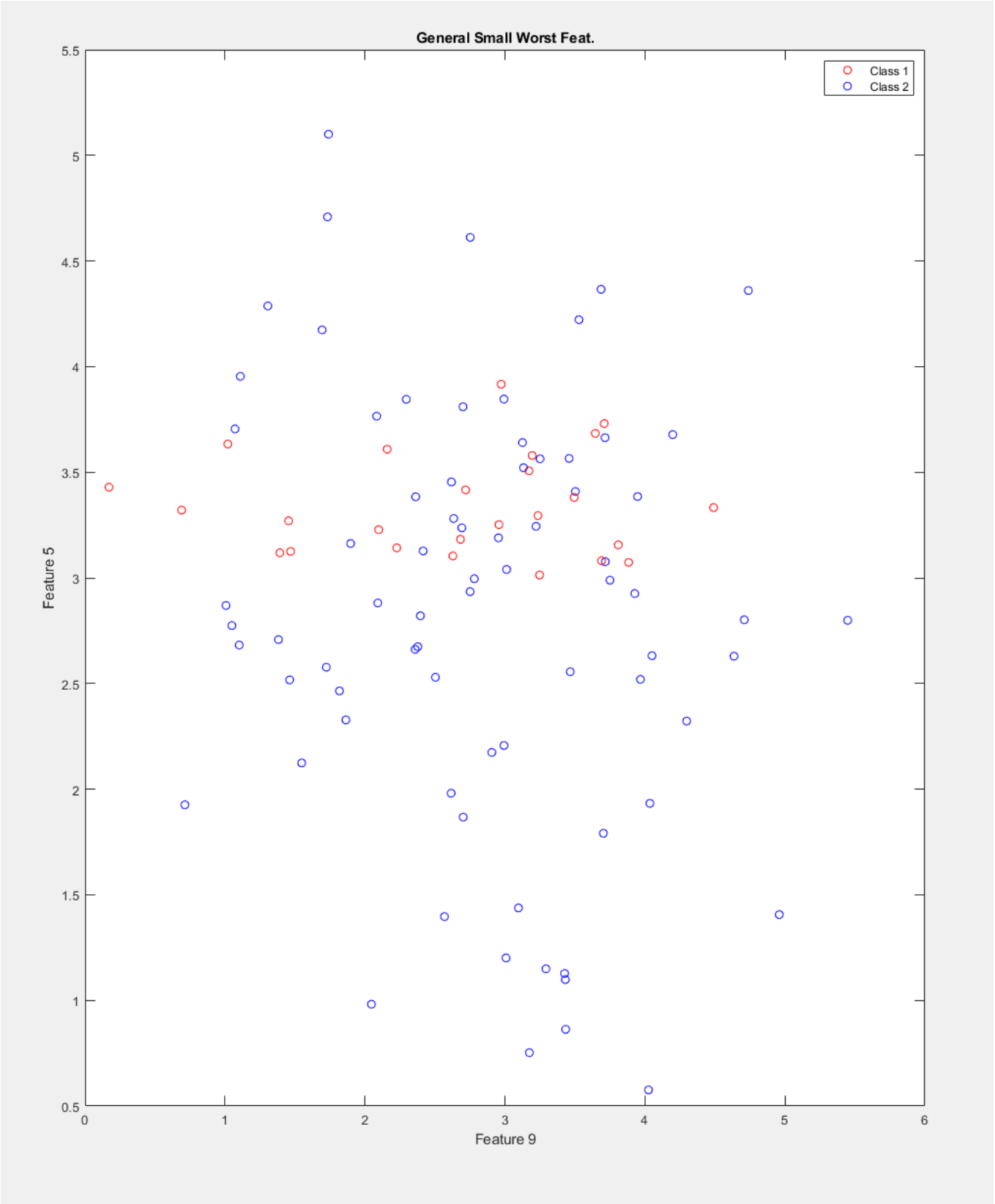
- **Two Plots: (Only two feature sets)**
  - **First Plot Best Features**
  - **Second Plot Worse Features**
  - **All plots Generated With MatLab, Code provided in zip**

General Small Sets:

Best: 92%

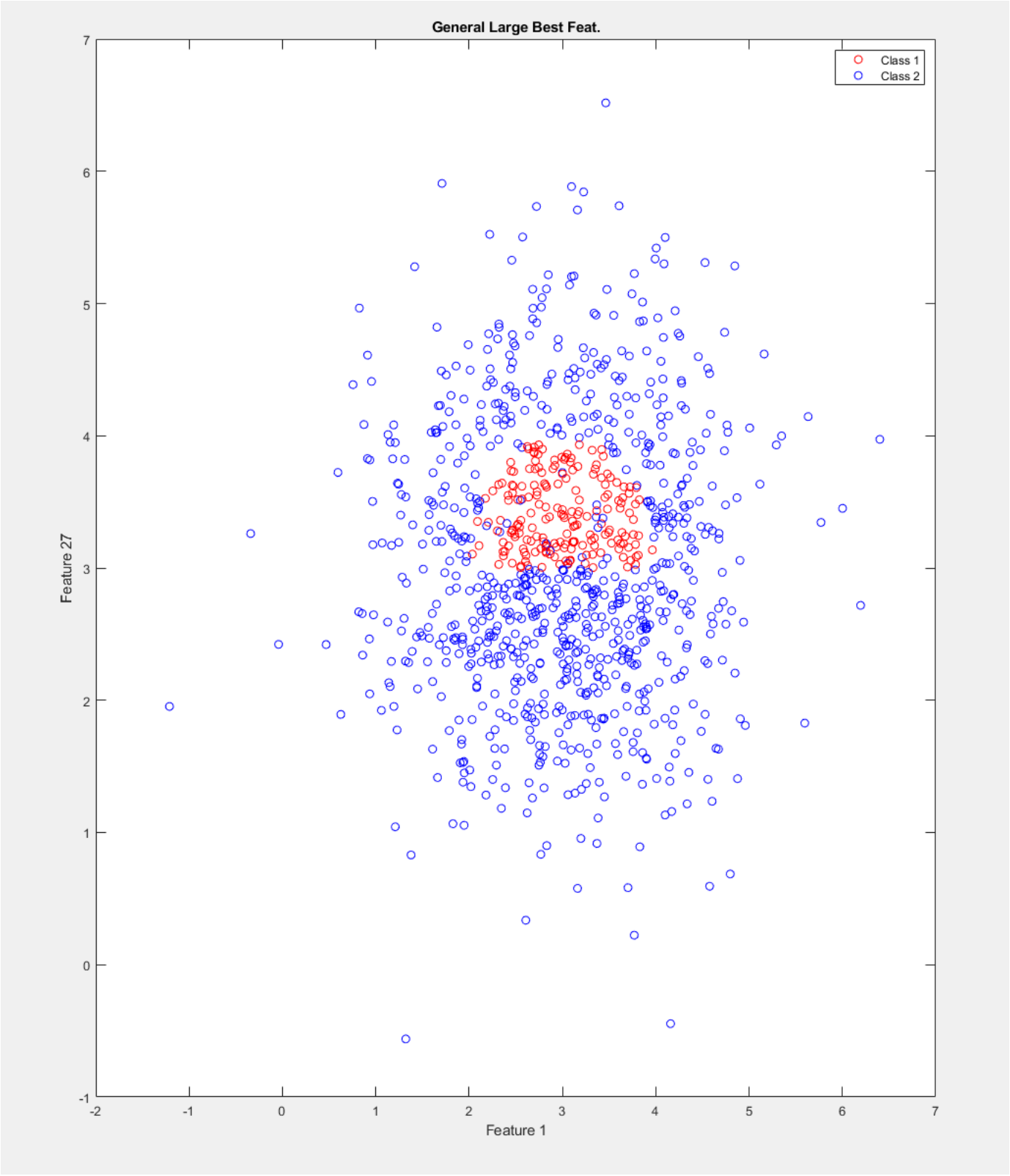


Worst: 73%

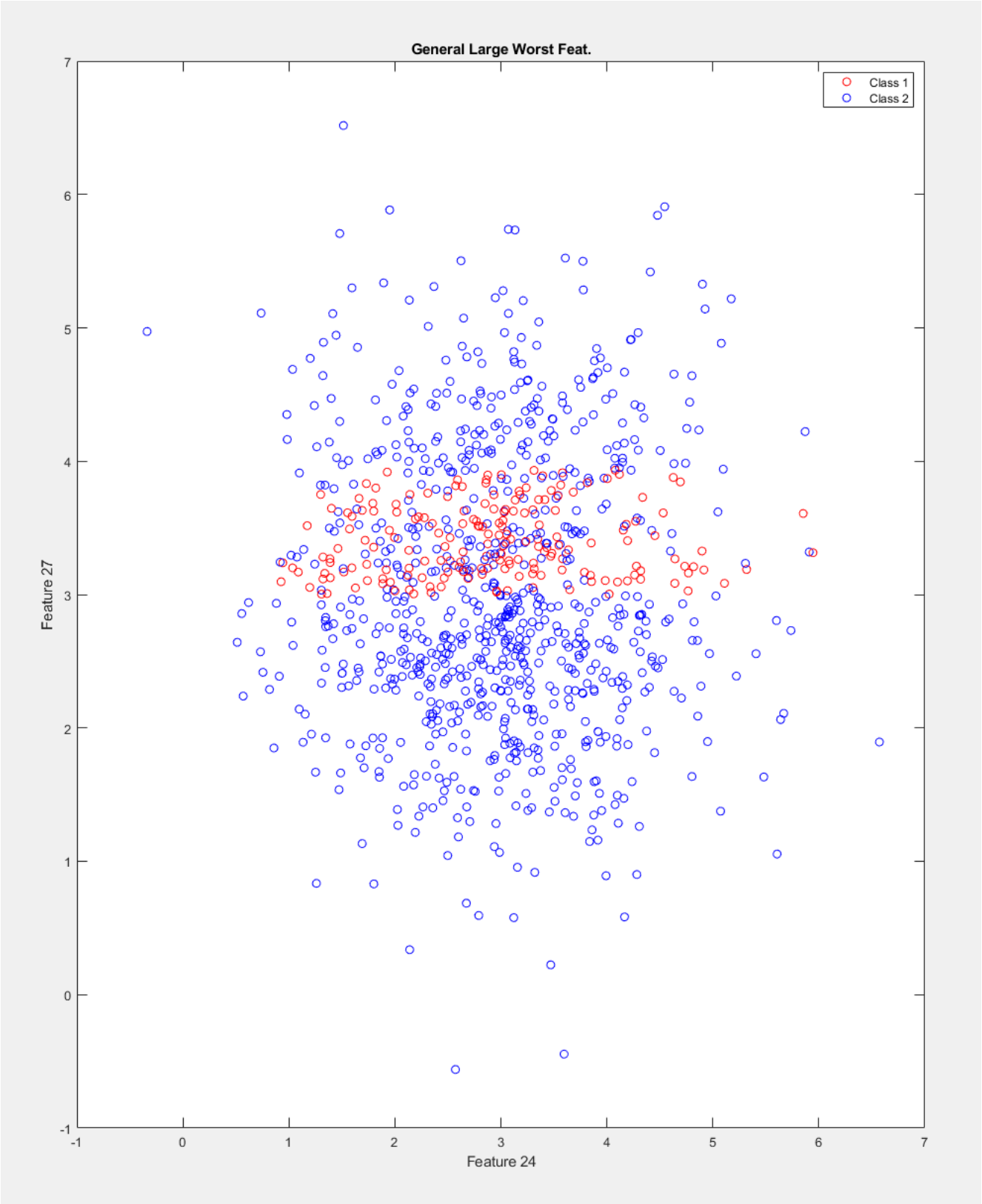

General Small Worst Feat.
Class 1
Class 2
Feature 5
Feature 9
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6

General Large Sets:

Best: 95.5%

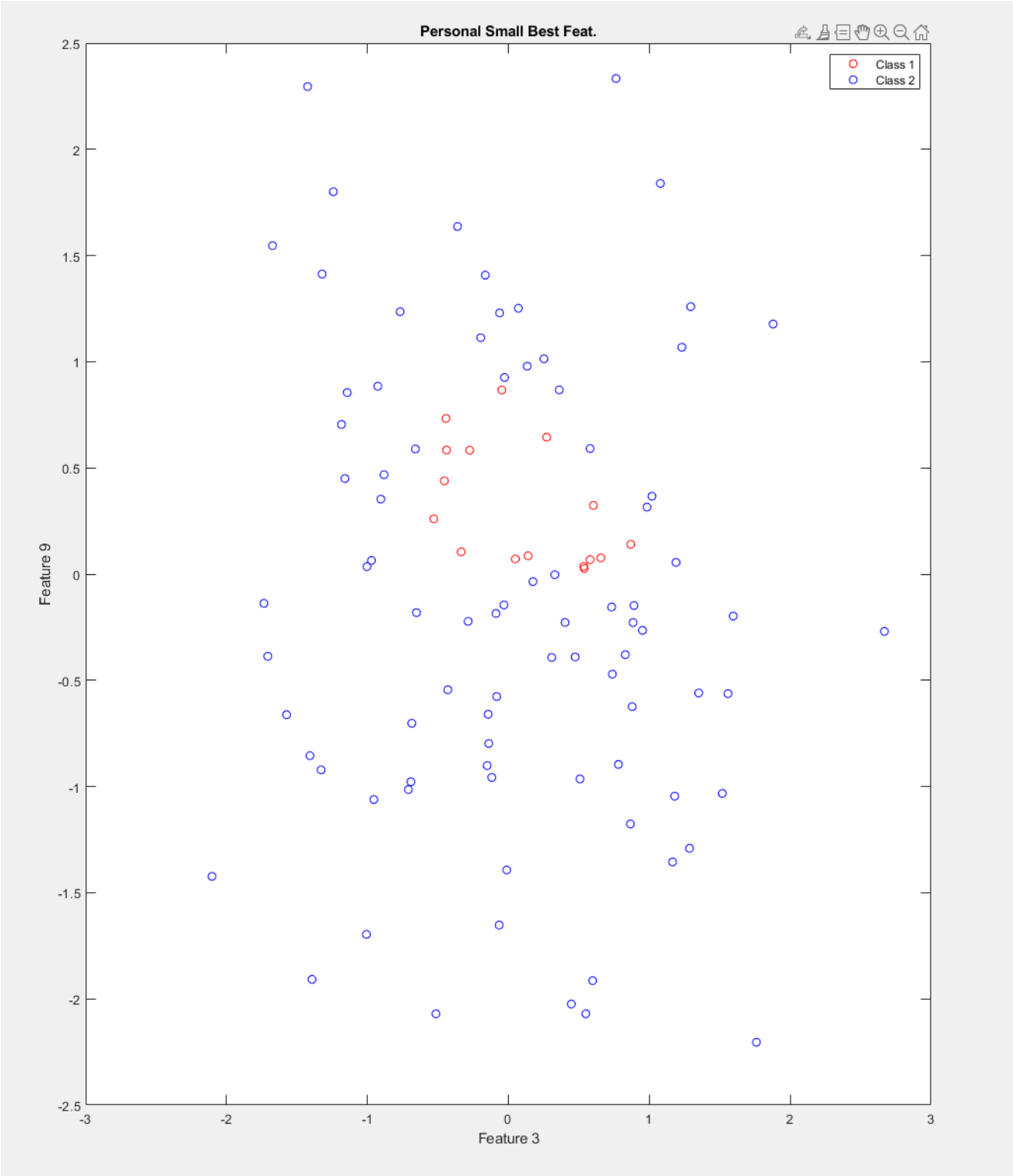


Worst: 81.2%

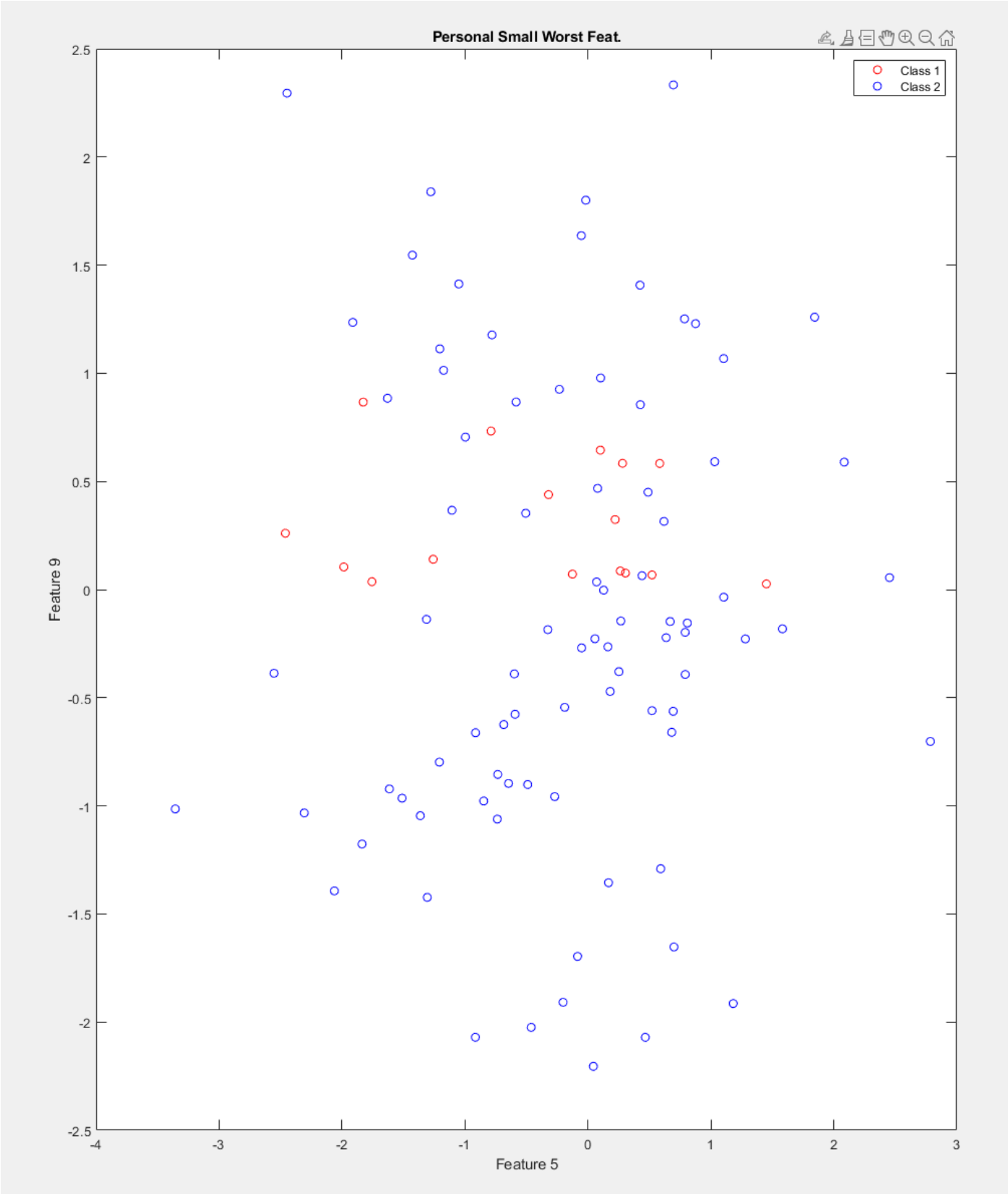

General Large Worst Feat.
Class 1
Class 2
Feature 27
Feature 24
7
6
5
4
3
2
1
0
-1
-1
0
1
2
3
4
5
6
7

Personal Small Sets:

Best: 93%

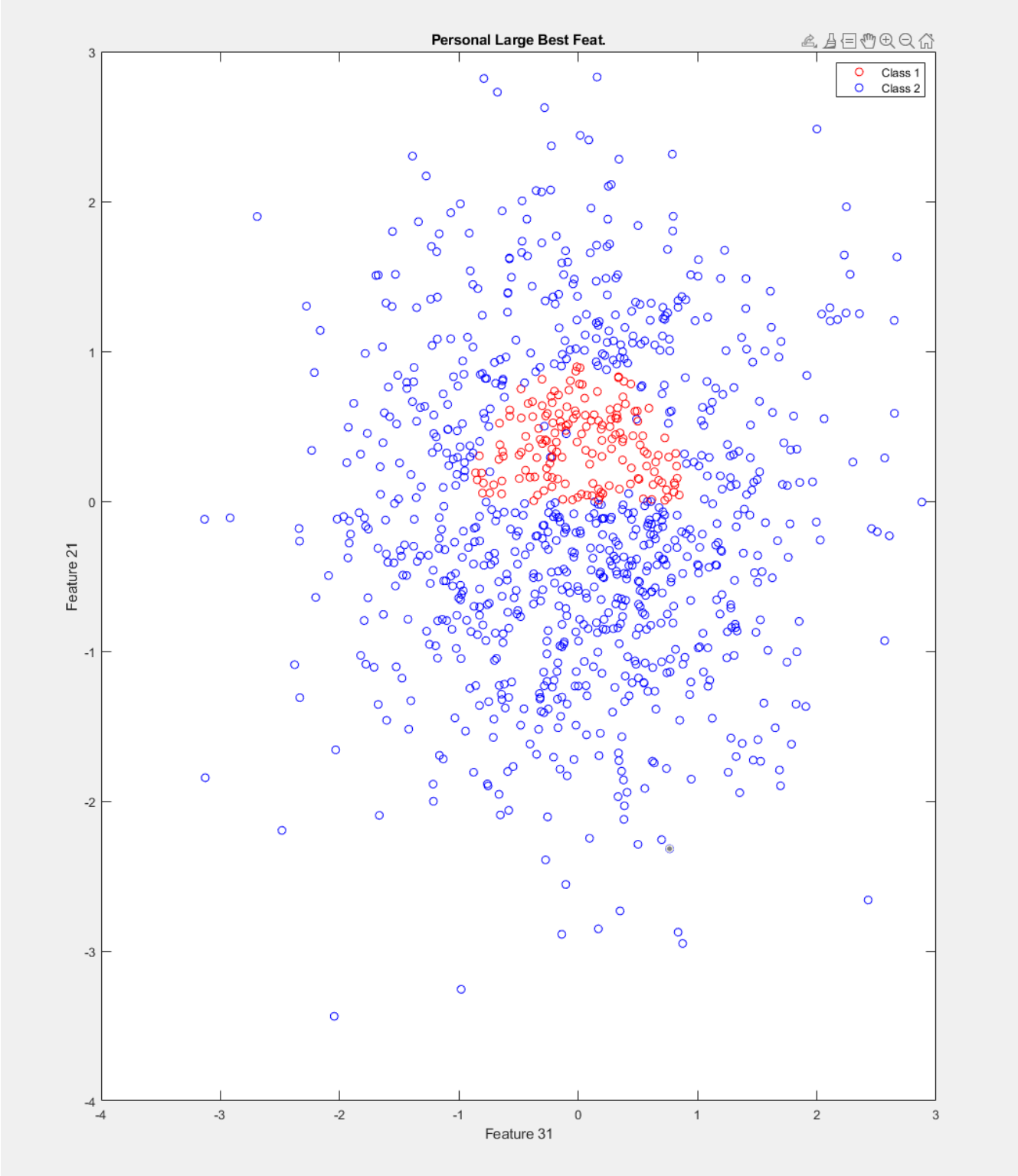


Worst: 75%

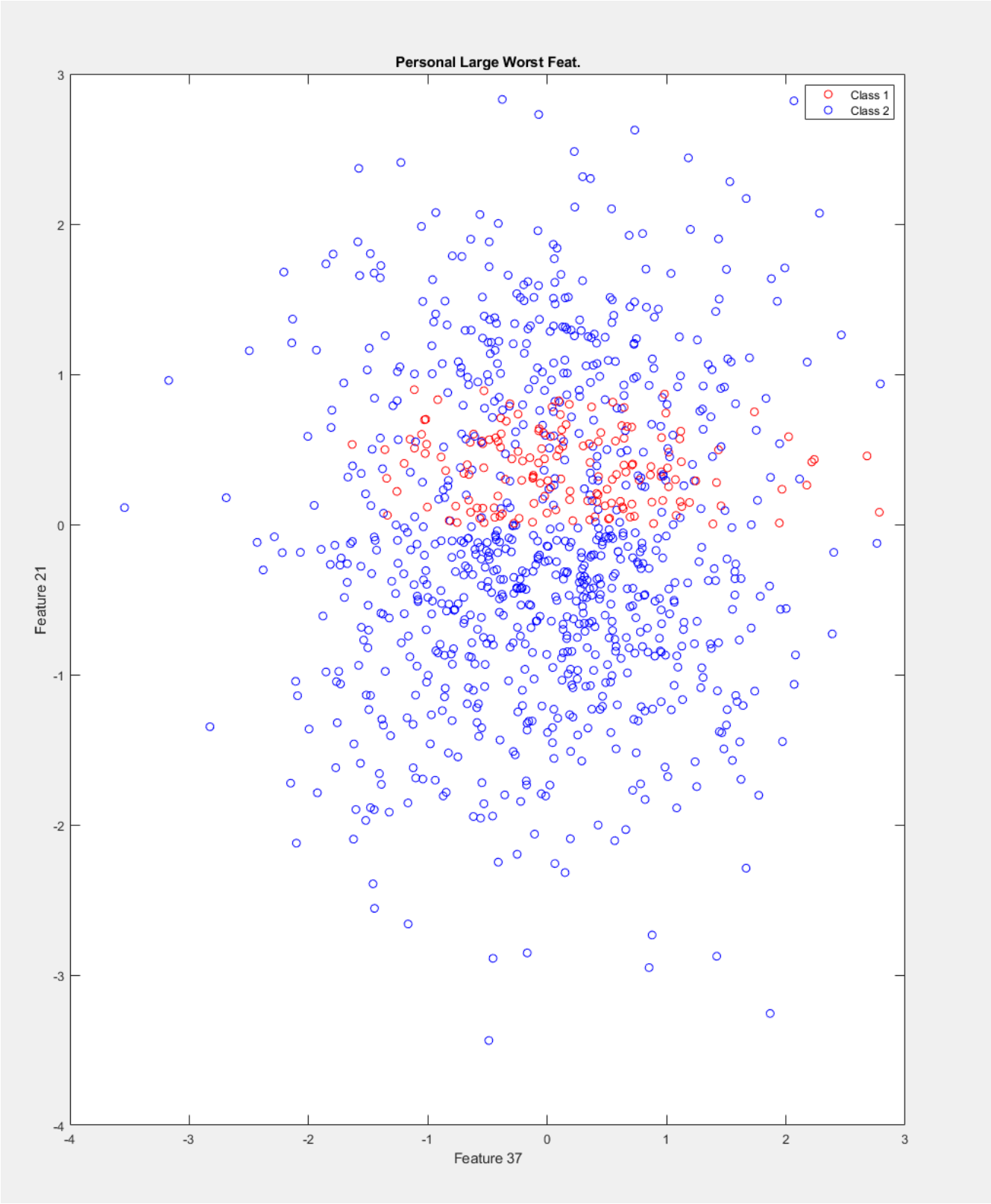

Personal Small Worst Feat.
Class 1
Class 2
Feature 9
Feature 5
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-4
-3
-2
-1
0
1
2
3

Personal Large Sets:

Best: 97.6%



Worst: 83%


Personal Large Worst Feat.
Class 1
Class 2
Feature 21
Feature 37
3
2
1
0
-1
-2
-3
-4
-4
-3
-2
-1
0
1
2
3

V. Algorithms

1. Forward Selection

- Will find the best feature subset that returns the highest accuracy from testing our Nearest Neighbor Classifier.
    - Performs a greedy search by iteratively adding one feature at a time and selecting the subset of highest accuracy. Repeat until accuracy decreases.

- Starts with an empty set of features and iterates through all unselected features.
    - Tests each possible subset of inserting one feature at a time, and finding the accuracy of our Nearest Neighbor Classifier using leave_one_out_cross_validation in our Validator class.
- Greedily add the feature that returned the highest overall accuracy to our feature set and repeat the process on the remaining unselected features.
- At each subsequent level of the search tree, we keep track of the best accuracy found so far and the feature subset that resulted in that.
    - If the best accuracy at a level decreases or we have inserted all features we stop, and return the best accuracy feature subset.

2. Backward Elimination

- Will find the best feature subset that returns the highest accuracy from testing our Nearest Neighbor Classifier.
    - Performs a greedy search by iteratively removing one feature at a time and selecting the subset of highest accuracy. Repeat until accuracy decreases.

- Starts with a full set of features and iterates through all unselected features.
    - Tests each possible subset of removing a single feature, finding the accuracy of our Nearest Neighbor Classifier using leave_one_out_cross_validation.
- Greedily removes the feature that resulted in the highest overall accuracy when removed from our current feature set.
- Keep track of the highest accuracy found thus far.
    - Repeat the above process on the remaining unselected features
        - If we find a feature subset with a higher accuracy, we update the highest accuracy feature subset found.
        - When our accuracy decreases or we have removed all possible features we stop.
    - Return the highest accuracy feature subset found.

VI. Analysis

Experiment 1: Comparing Forward Selection vs Backward Elimination

- Compare accuracy with no feature selection vs with feature selection
- Compare feature set and accuracy for forward selection vs backward elimination.
- Talk about pros and cons of both algorithms

Comparing accuracy with no feature selection (using all features) vs with feature selection

No Feature Selection (using all features)

Dataset	Accuracy
The General Small Dataset	68%
Our Personal Small Dataset (2)	66%
The General Large Dataset	69.4%
Our Personal Large Dataset (2)	72.3%

Feature Selection (using each search algorithm)

Dataset	Feature Selection Algorithm	Accuracy
The General Small Dataset	Forward Selection = {3,5}	92%
	Backward Elimination = {10,7,5,2,4}	83%
Our Personal Small Dataset (2)	Forward Selection = {3,9}	93%
	Backward Elimination = {10,9,7,2,1}	85%
The General Large Dataset	Forward Selection = {1,27}	95.5%
	Backward Elimination = {40,38,37,36,35,34,31,30,29,28,27,26,25,24,23,10,9,5,1,2,3,11,12,13,15,17,18,19,20,22}	74.5%
Our Personal Large Dataset (2)	Forward Selection = {31,21}	97.6%
	Backward Elimination = {40,39,38,37,36,35,34,33,30,29,27,26,25,24,23,10,9,8,6,5,1,2,3,4,11,12,13,14,15,16,17,18,19,20,21,22}	75.9%

- As shown in the tables above, the accuracy is around high 60 to low 70 percent when not using any feature selections. However, with the introduction of feature selection algorithms our accuracies increase by about 20-30% and are in the 80 to high 90 percent range.

Comparing feature set and accuracy for forward selection vs backward elimination

- For forward selection the feature sets obtained are much smaller at a size of 2. On the other hand, backward elimination obtains much larger feature sets (as a result of starting with the full size feature set) which range from a size of 5 for the small datasets and up to 30 for large datasets.
- For accuracy, all of our Forward Selection algorithms were able to obtain feature sets such that higher accuracies were found compared to those of the Backward Elimination algorithms.
    - The Forward Selection algorithm accuracies were all in the low to high 90 percent range.
    - The Backward Elimination algorithm had lower accuracies ranging from about 75 - 85 percent.
- Overall Forward Selection had better performance in terms of accuracy and required less features.

Pros/Cons

- **Pros of Forward Selection Search**
    - Fast runtime across both small and large datasets
    - Efficient for small and large datasets (is not as computationally intensive)
- **Cons of Forward Selection Search**
    - Can get trapped in local optima
        - Due to greedy search, our algorithm may stop after finding a decrease in accuracy at a specific level despite potentially better solutions existing further along the search tree.
    - Doesn't initially consider combinations of the majority of features that could provide greater distinctions of classes.

- **Pros of Backward Elimination Search**
    - Robustness
        - Due to training on a larger set of features, we may reduce overfitting and have a model that may become more accurate when generalized to other datasets
    - Can be efficient on datasets where combinations of the majority of features are useful for distinction of classes.
        - Starts with larger datasets first so that combination will be found quicker.
- **Cons of Backward Elimination Search**

- More computationally intensive (requires a lot more calculations due to working with larger sets of features)
- Longer runtimes across both small and large datasets
  - Large datasets drastically increase this due to the large increase in required computations at each iteration
- Can get trapped in local optima
  - Due to greedy search, our algorithm may stop after finding a decrease in accuracy at a specific level despite potentially better solutions existing further along the search tree.

VII. Conclusion

To conclude the project, we have found some differences in the method of implementation between the two given algorithms: Forward Selection and Backward Elimination. They are both useful when attempting to determine the best feature sets needed to accurately determine the classification of an object. While both are useful, one simply takes longer than the other. Backwards Elimination suffers in larger data sets as it has to continue through the different feature sets to find the most optimal solution. It takes a while to iterate through in comparison to the Forward Selection Algorithm. Additionally, the backwards elimination algorithm provides a larger subset which in turn is more data that we use up.

Unfortunately, the Backwards Elimination algorithm took too long to run and caused us a bit of time in attempting to figure out what the cause was. We found that in its attempt to go through all the sets, one by one, that it was too much for it to analyze. The data simply became too large for it to quickly go through and spit out a solution. To combat this in future development, we could add another data structure to help alleviate the strain on the system. Another solution can simply come with the hardware we are using, possibly slowing down the AI a bit.

The Forward Selection algorithm was fast and efficient so there is no optimization needed there at least with the current datasets we have. Otherwise, we could implement a hashmap or another data structure that will allow us to speed through and generate the features.

VIII. Trace of our small dataset

General Small Test Data Trace

```
Welcome to Group 3's CS170 Project 2 - Machine Learning Feature Selection Algorithm
Type in the name of the file to test: small-test-dataset.txt

Type the number of the algorithm you want to run.

        1) Forward Selection
        2) Backward Elimination


1
This dataset has 10 features (not including the class attribute), with 100 instances.

Calling Forward Selection Search

Running nearest neighbor with no features (default rate), using "leaving-one-out" evaluation, I get an accuracy of 75%

Beginning search

        Using feature(s) {1} accuracy is 57%
        Using feature(s) {2} accuracy is 54%
        Using feature(s) {3} accuracy is 68%
        Using feature(s) {4} accuracy is 65%
        Using feature(s) {5} accuracy is 75%
        Using feature(s) {6} accuracy is 61%
        Using feature(s) {7} accuracy is 62%
        Using feature(s) {8} accuracy is 60%
        Using feature(s) {9} accuracy is 66%
        Using feature(s) {10} accuracy is 64%

Feature set {5} was best, accuracy is 75%

        Using feature(s) {1,5} accuracy is 76%
        Using feature(s) {2,5} accuracy is 80%
        Using feature(s) {3,5} accuracy is 92%
        Using feature(s) {4,5} accuracy is 75%
        Using feature(s) {6,5} accuracy is 79%
        Using feature(s) {7,5} accuracy is 80%
        Using feature(s) {8,5} accuracy is 77%
        Using feature(s) {9,5} accuracy is 73%
        Using feature(s) {10,5} accuracy is 83%

Feature set {3,5} was best, accuracy is 92%

        Using feature(s) {1,3,5} accuracy is 84%
        Using feature(s) {2,3,5} accuracy is 79%
        Using feature(s) {4,3,5} accuracy is 84%
        Using feature(s) {6,3,5} accuracy is 82%
        Using feature(s) {7,3,5} accuracy is 89%
        Using feature(s) {8,3,5} accuracy is 79%
        Using feature(s) {9,3,5} accuracy is 83%
        Using feature(s) {10,3,5} accuracy is 87%

Feature set {7,3,5} was best, accuracy is 89%

(Warning, Accuracy has decreased!)
Finished Search!! The best features subset is {3,5}, which has an accuracy of 92%
```

Personal Validation Small Data Trace

```
Welcome to Group 3's CS170 Project 2 - Machine Learning Feature Selection Algorithm
Type in the name of the file to test: CS170_Spring_2023_Small_data__2.txt

Type the number of the algorithm you want to run.

        1) Forward Selection
        2) Backward Elimination


1
This dataset has 10 features (not including the class attribute), with 100 instances.

Calling Forward Selection Search

Running nearest neighbor with no features (default rate), using "leaving-one-out" evaluation, I get an accuracy of 84%

Beginning search

        Using feature(s) {1} accuracy is 76%
        Using feature(s) {2} accuracy is 73%
        Using feature(s) {3} accuracy is 77%
        Using feature(s) {4} accuracy is 73%
        Using feature(s) {5} accuracy is 71%
        Using feature(s) {6} accuracy is 71%
        Using feature(s) {7} accuracy is 74%
        Using feature(s) {8} accuracy is 75%
        Using feature(s) {9} accuracy is 84%
        Using feature(s) {10} accuracy is 78%

Feature set {9} was best, accuracy is 84%

        Using feature(s) {1,9} accuracy is 85%
        Using feature(s) {2,9} accuracy is 82%
        Using feature(s) {3,9} accuracy is 93%
        Using feature(s) {4,9} accuracy is 81%
        Using feature(s) {5,9} accuracy is 75%
        Using feature(s) {6,9} accuracy is 85%
        Using feature(s) {7,9} accuracy is 89%
        Using feature(s) {8,9} accuracy is 80%
        Using feature(s) {10,9} accuracy is 86%

Feature set {3,9} was best, accuracy is 93%

        Using feature(s) {1,3,9} accuracy is 88%
        Using feature(s) {2,3,9} accuracy is 86%
        Using feature(s) {4,3,9} accuracy is 83%
        Using feature(s) {5,3,9} accuracy is 86%
        Using feature(s) {6,3,9} accuracy is 81%
        Using feature(s) {7,3,9} accuracy is 86%
        Using feature(s) {8,3,9} accuracy is 83%
        Using feature(s) {10,3,9} accuracy is 87%

Feature set {1,3,9} was best, accuracy is 88%

(Warning, Accuracy has decreased!)
Finished Search!! The best features subset is {3,9}, which has an accuracy of 93%
```