Justin Luo
jluo036@ucr.edu
SID: 862013749
Due: 3/19/21

CS170: Introduction to Artificial Intelligence, Project 2
Dr. Eamonn Keogh

In completing this assignment I consulted:

- Dr. Eamonn Keogh's Matlab code
- Small and large dataset provided by Dr. Eamonn Keogh

Libraries used:

- <fstream> to read from file
- <vector> to store data
- <iostream> for cin, cout, and string streams
- <iomanip> to use setprecision()
- <math.h> to use sqrt(), pow(), and max()
- <algorithm> to use find()
- <limits.h> to use INT_MAX

Table of contents:

Report

The goal of this project is to find which subset of features yields the highest accuracy, thus would be best suited to use for classification. “Leave-one-out” cross validation and the nearest neighbor classifier were used together to measure the accuracy of a given subset. Two search algorithms, Forward Selection and Backwards Elimination, were tested on a small (10 features, 300 instances) and large (100 features, 500 instances) dataset. Forward Selection begins with an empty subset of features, adds one feature at a time, and keeps the feature that yields the highest accuracy. Backwards Elimination begins with a set of all features, removes one feature at a time, and keeps the subset that yields the highest accuracy. I was assigned the data files CS170_SMALLtestdata__46.txt and CS170_largetestdata__10.txt The results of these tests are shown in figures 1 through 5.

Figure 1 below shows the accuracies on the given subsets of features. The search algorithm used was Forward Selection and the data file tested was CS170_SMALLtestdata__46.txt.

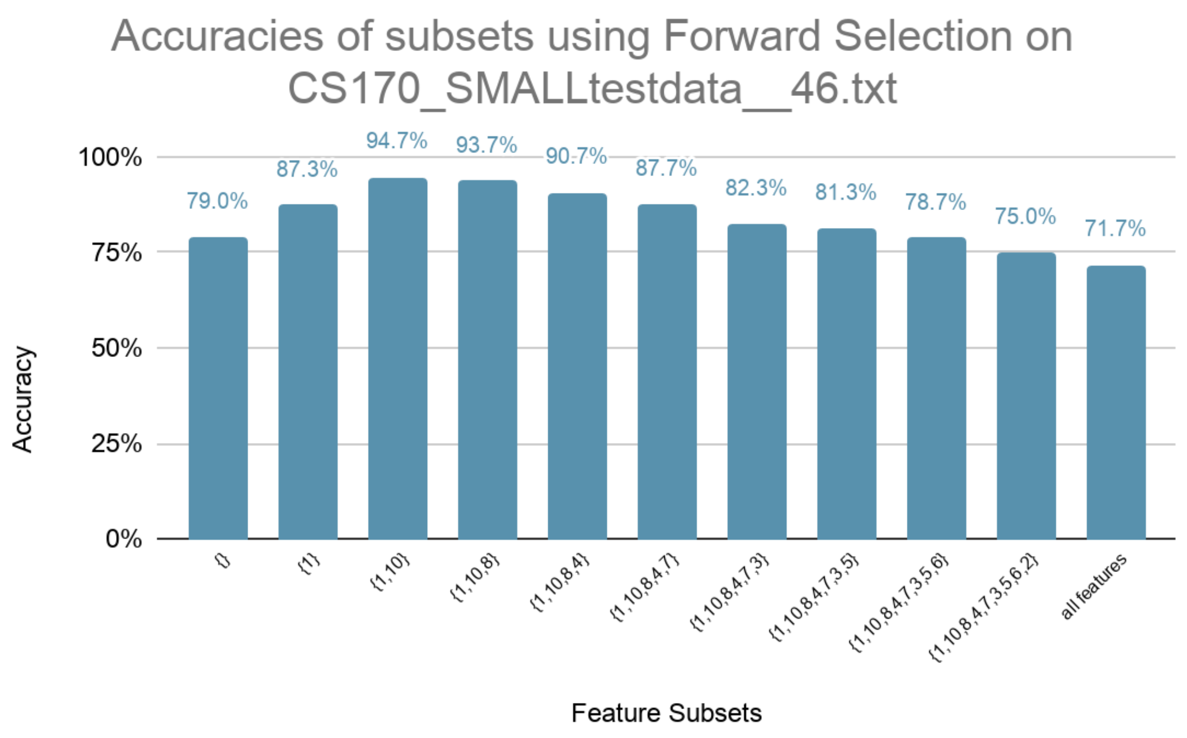


Figure 1: Accuracy of increasingly large subsets of features using Forward Selection on the file CS170_SMALLtestdata__46.txt

Starting from the default rate(the subset with no features), we see the accuracy is 79%. Adding feature 1 increases the accuracy to 87.3%. Adding feature 10 to feature 1 increases the accuracy to 94.7%. Adding features from here gradually decreases the accuracy. When all features are added, the accuracy is 71.7%.

Figure 2 below shows the accuracies on the given subsets of features using the same dataset, CS170_SMALLtestdata__46.txt. This time however, the Backwards Elimination algorithm was used.

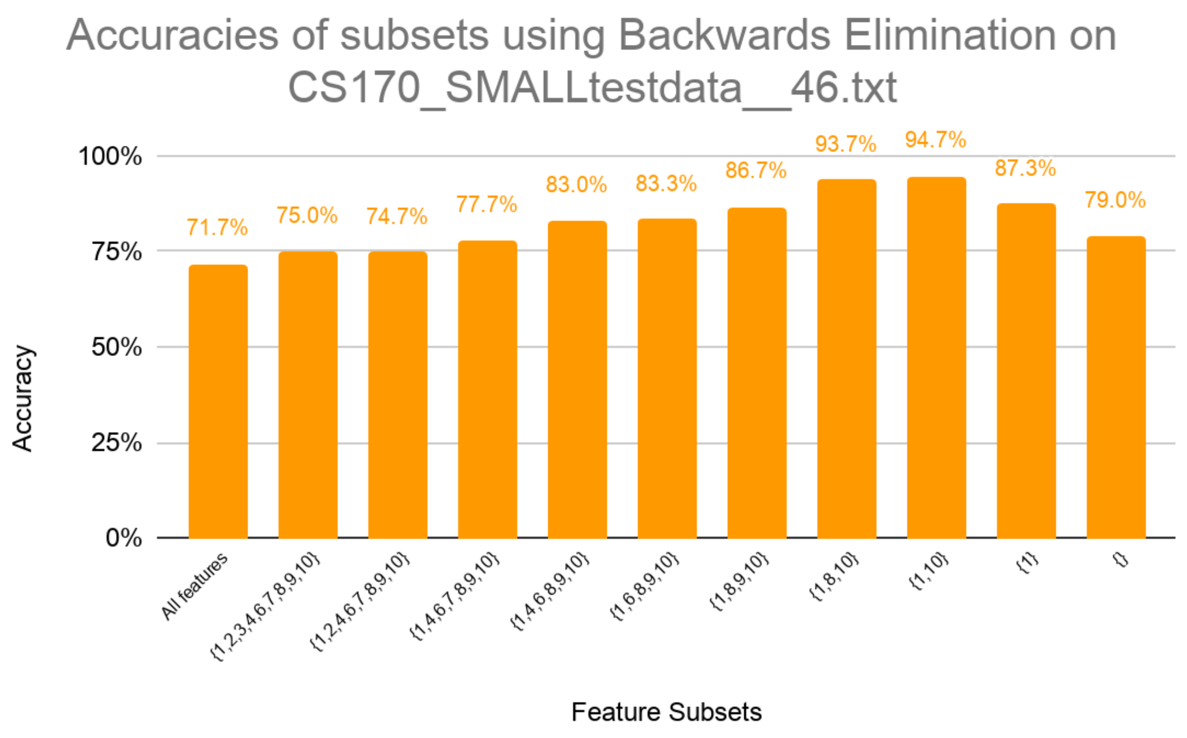


Figure 2: Accuracy of increasingly small subsets of features using Backwards Elimination on the file CS170_SMALLtestdata__46.txt

Starting with all features we have an accuracy of 71.7%. Removing feature 5 increases the accuracy to 75%. After that, removing feature 3 decreases the accuracy by 0.3%. The accuracy then increases as features are removed and peaks at 94.7% using features 1 and 10. The accuracy then decreases if any more features are removed.

Small dataset conclusion:
The combination of features 1 and 10 was reported to have the highest accuracy in both algorithms, thus these features are best suited to use to classify these two classes. Deploying a model using features 1 and 10, you should expect an accuracy of 94.7%.

Figure 3 below shows the accuracies on the given subsets of features. The search algorithm used was Forward Selection and the data file tested was CS170_largetestdata__10.txt.

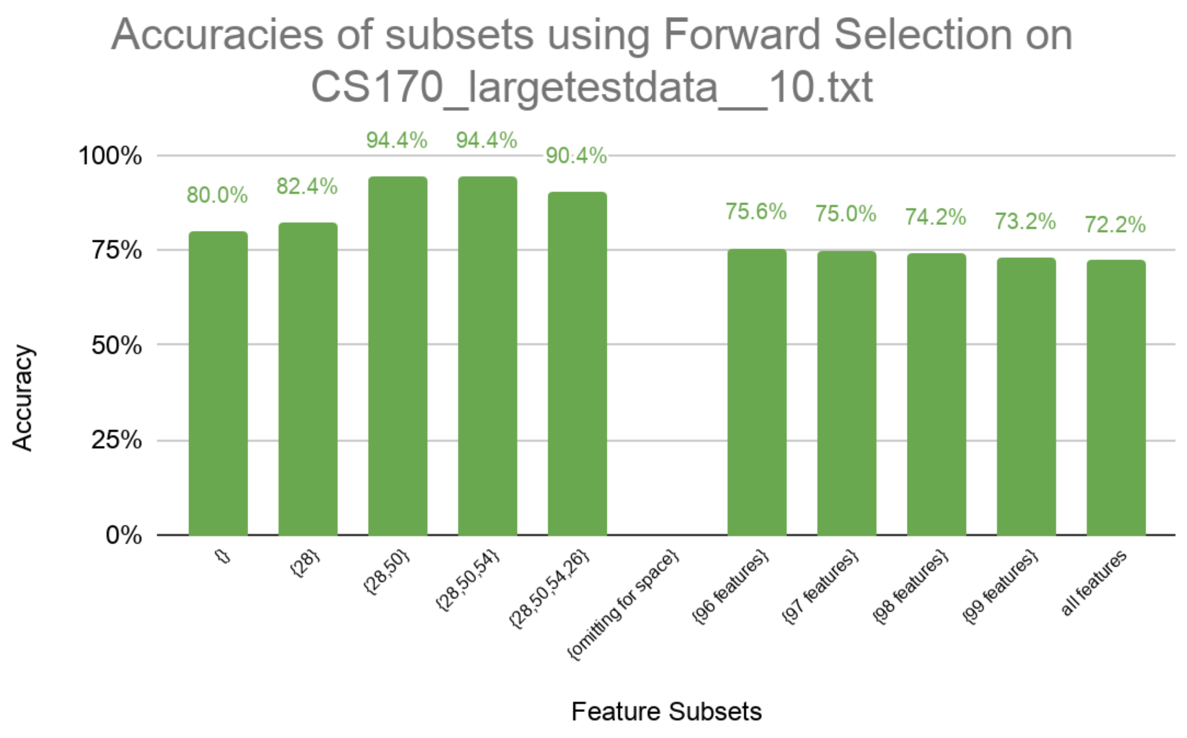


Figure 3: Accuracy of increasingly large subsets of features using Forward Selection on the file CS170_largetestdata__10.txt

Starting with no features, the accuracy is 80%. Adding feature 28 increases the accuracy to 82.4%. Adding feature 50 to feature 28 further increases the accuracy to 94.4%. Surprisingly, adding feature 54 to {28,50} does not change the accuracy at all. Adding feature 26 to that, however, does lower the accuracy by 4%. We see a trend that the last 5 subsets gradually decrease in accuracy as features are added. Using this info and the fact that there is a decrease in accuracy using 4 features from 3 features, we can infer that most, if not all, of the subsets omitted in the graph will decrease in accuracy as features are added. The highest accuracy is 94.4% and is obtained using features {28,50} and {28,50,54}.

Figure 4 below shows the accuracies on the given subsets of features using the same dataset, CS170_largetestdata__10.txt. This time however, the Backwards Elimination algorithm was used.

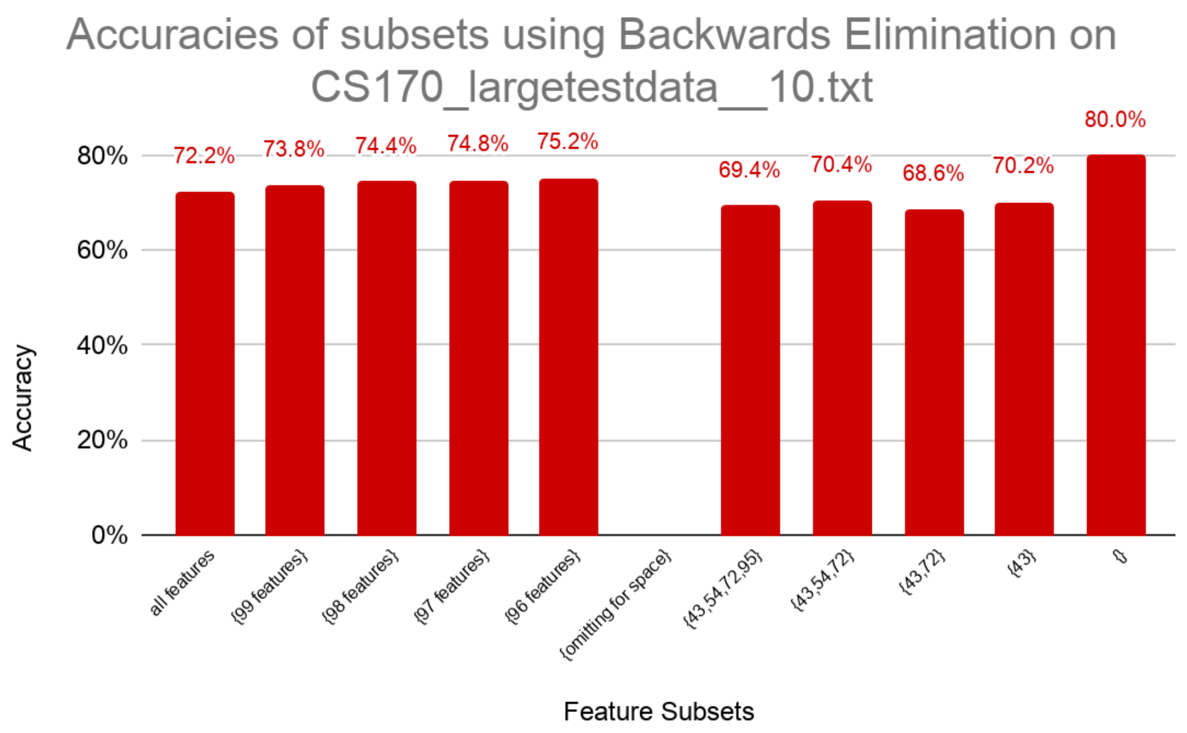


Figure 4: Accuracy of increasingly small subsets of features using Backwards Elimination on the file CS170_largetestdata__10.txt

Running Backwards Elimination on the large data set yielded poor results compared to Forwards Selection. We see in the first 5 bars of the graph that the accuracy steadily increases as features are removed. However, the last few bars of the graph show unexpected results. Using 1,2, or 3 features on the Backwards Elimination algorithm we see that the accuracies are 70.2%, 68.6%, and 70.4% respectively. These accuracies are much lower than if we had used 1,2, or 3 features on the Forward Selection algorithm, which give accuracies of 82.4%, 94.4%, and 94.4% respectively. The highest accuracy reported using Backwards Elimination was 81.4%, found on subset:

{1,4,6,7,8,11,12,18,19,22,24,26,27,28,30,33,36,37,38,39,40,41,43,44,49,51,52,54,58,59,61,64, 65,66,69,70,72,73,75,76,77,78,79,84,85,86,87,89,91,92,94,95,96} (not depicted in figure 4)

This is 13% lower than the highest accuracy found using Forward Selection (94.4%) and only 1.4% better than the default rate (80%).

Large dataset conclusion:
Forward Selection outperformed Backwards Elimination when testing the large dataset. The highest accuracy found using Forward Selection was 94.4% at a subset of 2 and 3 features. The highest accuracy found using Backwards Elimination was only 81.4%, just 1.4% above the default rate, at a subset of 53 features. The subset {28,50} or {28,50,54} gives the highest accuracy. However, since feature 54 does not change the accuracy at all, it may not be a useful feature, thus I would leave it out. The best set of features to use in a model is {28,50}; you should expect an accuracy of 94.4% using these features.

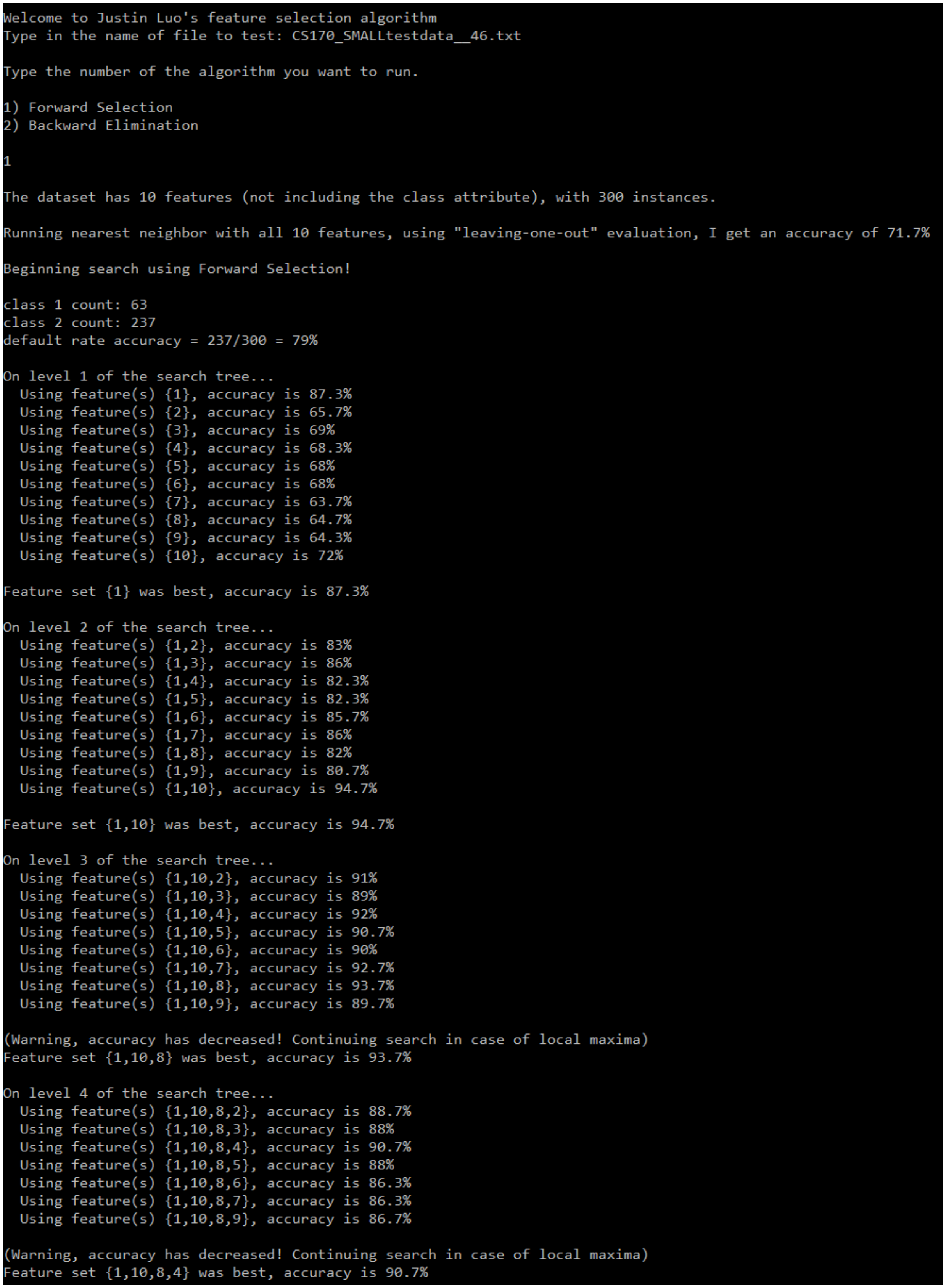
Search runtime: I implemented the search using C++. Running on a computer with an AMD Ryzen 5 3600 and 16 gigs of RAM, I got the following runtimes using unix's time command.

	Small dataset (10 features, 300 instances)	Large dataset (100 features, 500 instances)
Forward Selection	1.1 sec	39 min 44 sec
Backward Elimination	1.9 sec	76 min 36 sec

Figure 5: table depicting runtimes

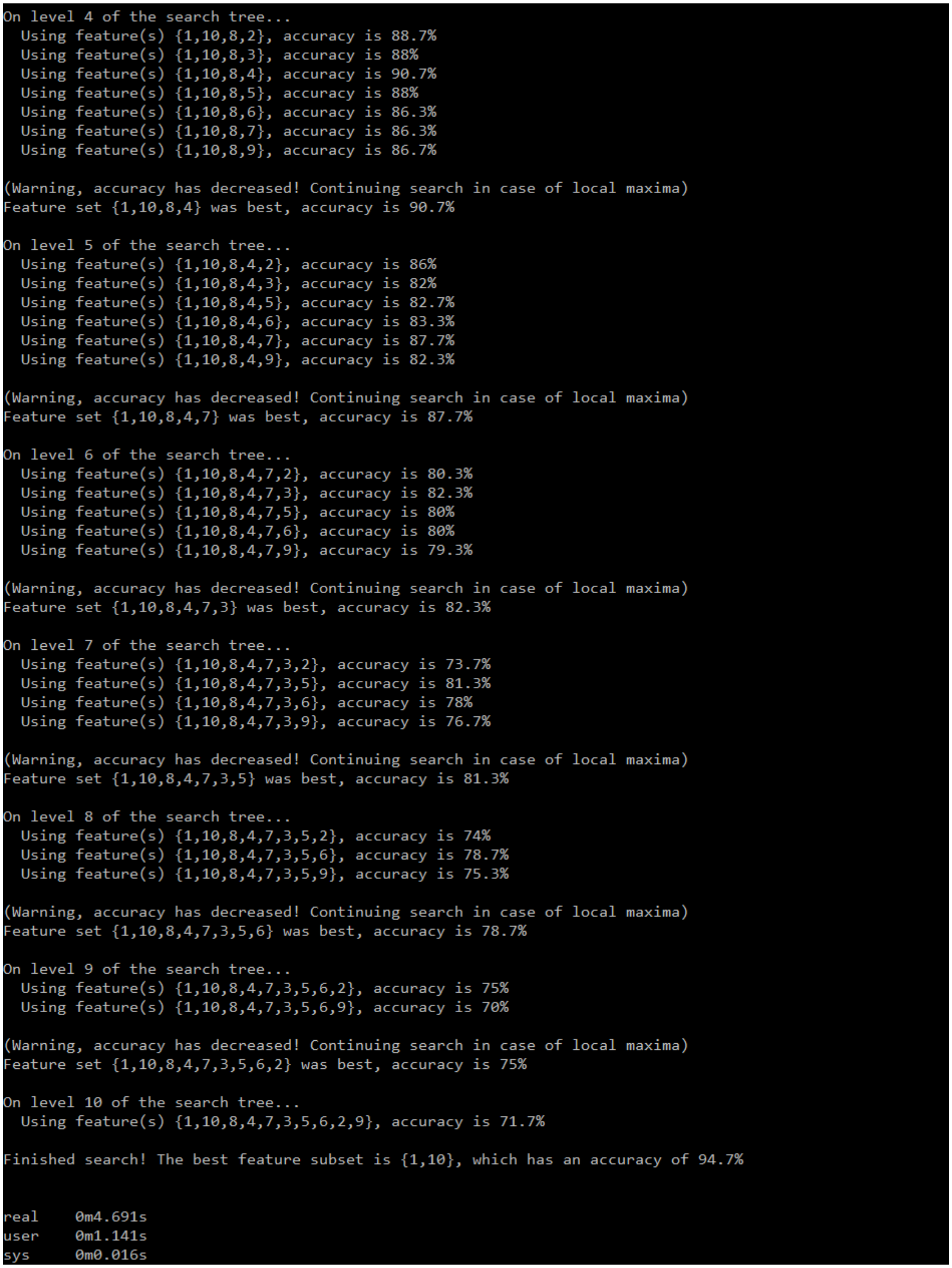
From figure 5, we see that In both the small and large dataset, Forward Selection ran significantly faster than Backward Elimination. This is to be expected, since in Forward Selection, the size of a subset to test decreases as the number of subsets to test increases, whereas in Backwards Elimination, the size of a subset to test increases as the number of subsets to test increases.

Trace of forward Selection on small dataset



```
Welcome to Justin Luo's feature selection algorithm
Type in the name of file to test: CS170_SMALLtestdata__46.txt

Type the number of the algorithm you want to run.

1) Forward Selection
2) Backward Elimination

1

The dataset has 10 features (not including the class attribute), with 300 instances.

Running nearest neighbor with all 10 features, using "leaving-one-out" evaluation, I get an accuracy of 71.7%

Beginning search using Forward Selection!

class 1 count: 63
class 2 count: 237
default rate accuracy = 237/300 = 79%

On level 1 of the search tree...
  Using feature(s) {1}, accuracy is 87.3%
  Using feature(s) {2}, accuracy is 65.7%
  Using feature(s) {3}, accuracy is 69%
  Using feature(s) {4}, accuracy is 68.3%
  Using feature(s) {5}, accuracy is 68%
  Using feature(s) {6}, accuracy is 68%
  Using feature(s) {7}, accuracy is 63.7%
  Using feature(s) {8}, accuracy is 64.7%
  Using feature(s) {9}, accuracy is 64.3%
  Using feature(s) {10}, accuracy is 72%

Feature set {1} was best, accuracy is 87.3%

On level 2 of the search tree...
  Using feature(s) {1,2}, accuracy is 83%
  Using feature(s) {1,3}, accuracy is 86%
  Using feature(s) {1,4}, accuracy is 82.3%
  Using feature(s) {1,5}, accuracy is 82.3%
  Using feature(s) {1,6}, accuracy is 85.7%
  Using feature(s) {1,7}, accuracy is 86%
  Using feature(s) {1,8}, accuracy is 82%
  Using feature(s) {1,9}, accuracy is 80.7%
  Using feature(s) {1,10}, accuracy is 94.7%

Feature set {1,10} was best, accuracy is 94.7%

On level 3 of the search tree...
  Using feature(s) {1,10,2}, accuracy is 91%
  Using feature(s) {1,10,3}, accuracy is 89%
  Using feature(s) {1,10,4}, accuracy is 92%
  Using feature(s) {1,10,5}, accuracy is 90.7%
  Using feature(s) {1,10,6}, accuracy is 90%
  Using feature(s) {1,10,7}, accuracy is 92.7%
  Using feature(s) {1,10,8}, accuracy is 93.7%
  Using feature(s) {1,10,9}, accuracy is 89.7%

(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {1,10,8} was best, accuracy is 93.7%

On level 4 of the search tree...
  Using feature(s) {1,10,8,2}, accuracy is 88.7%
  Using feature(s) {1,10,8,3}, accuracy is 88%
  Using feature(s) {1,10,8,4}, accuracy is 90.7%
  Using feature(s) {1,10,8,5}, accuracy is 88%
  Using feature(s) {1,10,8,6}, accuracy is 86.3%
  Using feature(s) {1,10,8,7}, accuracy is 86.3%
  Using feature(s) {1,10,8,9}, accuracy is 86.7%

(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {1,10,8,4} was best, accuracy is 90.7%
```

Trace of Forward Selection on small dataset, continued



```
On level 4 of the search tree...
  Using feature(s) {1,10,8,2}, accuracy is 88.7%
  Using feature(s) {1,10,8,3}, accuracy is 88%
  Using feature(s) {1,10,8,4}, accuracy is 90.7%
  Using feature(s) {1,10,8,5}, accuracy is 88%
  Using feature(s) {1,10,8,6}, accuracy is 86.3%
  Using feature(s) {1,10,8,7}, accuracy is 86.3%
  Using feature(s) {1,10,8,9}, accuracy is 86.7%

(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {1,10,8,4} was best, accuracy is 90.7%

On level 5 of the search tree...
  Using feature(s) {1,10,8,4,2}, accuracy is 86%
  Using feature(s) {1,10,8,4,3}, accuracy is 82%
  Using feature(s) {1,10,8,4,5}, accuracy is 82.7%
  Using feature(s) {1,10,8,4,6}, accuracy is 83.3%
  Using feature(s) {1,10,8,4,7}, accuracy is 87.7%
  Using feature(s) {1,10,8,4,9}, accuracy is 82.3%

(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {1,10,8,4,7} was best, accuracy is 87.7%

On level 6 of the search tree...
  Using feature(s) {1,10,8,4,7,2}, accuracy is 80.3%
  Using feature(s) {1,10,8,4,7,3}, accuracy is 82.3%
  Using feature(s) {1,10,8,4,7,5}, accuracy is 80%
  Using feature(s) {1,10,8,4,7,6}, accuracy is 80%
  Using feature(s) {1,10,8,4,7,9}, accuracy is 79.3%

(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {1,10,8,4,7,3} was best, accuracy is 82.3%

On level 7 of the search tree...
  Using feature(s) {1,10,8,4,7,3,2}, accuracy is 73.7%
  Using feature(s) {1,10,8,4,7,3,5}, accuracy is 81.3%
  Using feature(s) {1,10,8,4,7,3,6}, accuracy is 78%
  Using feature(s) {1,10,8,4,7,3,9}, accuracy is 76.7%

(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {1,10,8,4,7,3,5} was best, accuracy is 81.3%

On level 8 of the search tree...
  Using feature(s) {1,10,8,4,7,3,5,2}, accuracy is 74%
  Using feature(s) {1,10,8,4,7,3,5,6}, accuracy is 78.7%
  Using feature(s) {1,10,8,4,7,3,5,9}, accuracy is 75.3%

(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {1,10,8,4,7,3,5,6} was best, accuracy is 78.7%

On level 9 of the search tree...
  Using feature(s) {1,10,8,4,7,3,5,6,2}, accuracy is 75%
  Using feature(s) {1,10,8,4,7,3,5,6,9}, accuracy is 70%

(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {1,10,8,4,7,3,5,6,2} was best, accuracy is 75%

On level 10 of the search tree...
  Using feature(s) {1,10,8,4,7,3,5,6,2,9}, accuracy is 71.7%

Finished search! The best feature subset is {1,10}, which has an accuracy of 94.7%


real    0m4.691s
user    0m1.141s
sys     0m0.016s
```

featureSelection.cpp

URL: https://github.com/justinhluo/CS170

```
//Justin Luo - 862013749
#include <iostream>
#include <fstream>
#include <vector>
#include <iomanip>
#include <math.h>
#include <limits.h>
#include <algorithm>
using namespace std;

vector<vector<double>> readData() { //prompts user to enter file to read from and returns the data as a 3d array
	string filename;
	string str;
	vector<vector<double>> data;

	cout << endl << "Welcome to Justin Luo's feature selection algorithm "<< endl;
	cout << "Type in the name of file to test: ";

	while(1) { //loop until a file can be opened
		cin >> filename;
		ifstream inputfile(filename);
		if(!inputfile.is_open()) {
			cout << "Could not open file " << filename << ". Enter a differnt file: ";
		}else {
			while(getline(inputfile, str)){ //read line by line
				stringstream ss(str);
				double input;
				vector<double> line;
				while(ss >> input) { //read in each string of every line as a double
					line.push_back(input);
				}
				data.push_back(line); //push line vector onto data vector
			}
			inputfile.close();
			return data;
		}
	}
}

void printdata(vector<vector<double>> data) { //prints data(for testing)
	for(int i = 0; i < data.size(); ++i){
		for (int j = 0; j< data.at(i).size(); ++j){
			cout << setprecision(8) << data.at(i).at(j) << " ";
		}
		cout << endl;
	}
}

double crossValidate(vector<vector<double>> data, vector <int> curr_features, int feature_to_add, int addFeat) {
	double accuracy = 0;
	int num_correctly_classified = 0;
	for (int i = 0; i < data.size(); ++i) {

		int label = data.at(i).at(0);

		vector <double> object_to_classify = data.at(i);

		double nearest_neighbor_distance = INT_MAX;
```

```
				int nearest_neighbor_location = INT_MAX;

				int nearest_neighbor_label;

				for(int k = 0; k < data.size(); ++k) {
					if(k != i) { //don't compare object with itself
						double add_feat_dist = pow(object_to_classify.at(feature_to_add) -
data.at(k).at(feature_to_add), 2);
						double min_distance = 0;
						for (int j = 0; j < curr_features.size(); ++j) {
							min_distance += pow((object_to_classify.at(curr_features.at(j)) -
data.at(k).at(curr_features.at(j))), 2);
						}
						if(addFeat) {
							min_distance += add_feat_dist;
						}
						min_distance = sqrt(min_distance);
						if(min_distance < nearest_neighbor_distance) {
							nearest_neighbor_distance = min_distance;
							nearest_neighbor_location = k;
							nearest_neighbor_label = data.at(nearest_neighbor_location).at(0);
						}
					}
				}
				if (label == nearest_neighbor_label) {
					num_correctly_classified++;
				}
			}
			accuracy = (double)num_correctly_classified / (double)data.size();
			return accuracy;
}

void forwardSelection(vector<vector<double>> data) { //start with no features and add one by one
			cout << "Beginning search using Forward Selection!" << endl << endl;
			vector <int> curr_features; //stores the indices of added features
			vector <int> final_features;
			double final_accuracy = 0;
			double defaultrate;
			double class1cnt = 0;
			double class2cnt = 0;
			for (int i = 0; i < data.size(); ++i) {
				if(data.at(i).at(0) == 1) {
					class1cnt++;
				}else if(data.at(i).at(0) == 2) {
					class2cnt++;
				}
			}

			if (class1cnt > class2cnt){
				defaultrate = class1cnt/data.size();
			}else{
				defaultrate = class2cnt/data.size();
			}
			cout << "class 1 count: " <<  class1cnt <<endl;
			cout << "class 2 count: " <<  class2cnt <<endl;
			cout << "default rate accuracy = " << max(class1cnt,class2cnt) << "/" <<  data.size() << " = " << setprecision(3) <<
defaultrate * 100 << "%" << endl << endl;

			for(int i = 1; i < data.at(0).size(); ++i) {
				cout << "On level " << i << " of the search tree..." << endl;
```

```
        int feature_to_add;
        double best_accuracy = 0;

        for (int  k = 1; k < data.at(0).size(); ++k) {

            if(find(curr_features.begin(), curr_features.end(), k) == curr_features.end()) { //feature at index k is not
in curr_features so we can add it

                cout << "  Using feature(s) {";
                for(int j = 0; j < curr_features.size(); ++j) { //will be empty first itereation, so just print k
                    cout << curr_features.at(j) << ",";
                }
                cout << k << "}, ";
                double accuracy = crossValidate(data, curr_features, k, 1);
                cout << "accuracy is " << setprecision(3) << accuracy * 100 << "%" << endl;
                if (accuracy > best_accuracy) {
                    best_accuracy = accuracy;
                    feature_to_add = k;
                }
            }
        }
        curr_features.push_back(feature_to_add);
        if(i != data.at(0).size() - 1){ //only print this stuff if not on last iteration
            if(best_accuracy >= final_accuracy) { //final_accuracy = max(best_accuracy)
                final_accuracy = best_accuracy;
                final_features = curr_features;
            }else {
                cout << endl << "(Warning, accuracy has decreased! Continuing search in case of local
maxima)";
            }
            cout << endl << "Feature set {";
            for(int j = 0; j < curr_features.size(); ++j) {
                if(j == curr_features.size() - 1) {
                    cout << curr_features.at(j);
                }else {
                    cout << curr_features.at(j) << ",";
                }
            }
            cout << "} was best, accuracy is " << setprecision(3) << best_accuracy * 100 << "%" << endl << endl;
        }
    }
    cout << endl << "Finished search! The best feature subset is {";
    for(int i = 0; i < final_features.size(); ++i){
        if(i == final_features.size() - 1) {
            cout << final_features.at(i);
        }else{
            cout << final_features.at(i) << ",";
        }
    }
    cout << "}, which has an accuracy of " << setprecision(3) << final_accuracy * 100 << "%" << endl << endl;
}

void backwardElimination(vector<vector<double>> data) { //start with all features and remove one by one
    double defaultrate;
    double class1cnt = 0;
    double class2cnt = 0;
    for (int i = 0; i < data.size(); ++i) {
        if(data.at(i).at(0) == 1) {
            class1cnt++;
        }else if(data.at(i).at(0) == 2) {
            class2cnt++;
        }
    }

    if (class1cnt > class2cnt){
        defaultrate = class1cnt/data.size();
    }else{
        defaultrate = class2cnt/data.size();
    }
    cout << "class 1 count: " <<  class1cnt <<endl;
    cout << "class 2 count: " <<  class2cnt <<endl;
    cout << "default rate accuracy = " << max(class1cnt,class2cnt) << "/" <<  data.size() << " = " << setprecision(3) <<
defaultrate * 100 << "%" << endl << endl;
    cout << "Beginning search using Backwards Elimination!" << endl << endl;
    vector <int> curr_features;
    for(int i = 1; i < data.at(0).size(); ++i) { //push all features onto curr_features
        curr_features.push_back(i);
    }
    vector <int> final_features;
    double final_accuracy = 0;
    cout << "On level 1 of the search tree..." << endl; //hardcoding first iteration of loop
    cout << " Using feature(s) {";
    for(int i = 0; i < curr_features.size(); ++i) {
        if(i == curr_features.size() - 1) {
            cout << curr_features.at(i);
        }else {
            cout << curr_features.at(i) << ",";
        }
    }
    cout << "}, ";
    double accuracy = crossValidate(data, curr_features, curr_features.size(), 0);
    cout << "accuracy is " << setprecision(3) << accuracy * 100 << "%" << endl << endl;

    for(int i = 2; i < data.at(0).size(); ++i) {
        cout << "On level " << i << " of the search tree..." << endl;
        int feature_to_remove;
        double best_accuracy = 0;

        for(int k = 0; k < curr_features.size(); ++k) {
            vector<int>remove_one = curr_features; //testing removing one index at a time to find highest
accuracy
            remove_one.erase(remove_one.begin() + k); //erase kth index of curr_features
            cout << "  Using feature(s) {";
            for(int j = 0; j < remove_one.size(); ++j) {
                if(j == remove_one.size() - 1) {
                    cout << remove_one.at(j);
                }else {
                    cout << remove_one.at(j) << ",";
                }
            }
            cout << "}, ";
            double accuracy = crossValidate(data, remove_one, k, 0);
            cout << "accuracy is " << setprecision(3) << accuracy * 100 << "%" << endl;
            if (accuracy > best_accuracy) {
                best_accuracy = accuracy;
                feature_to_remove = k;
            }
        }
        curr_features.erase(curr_features.begin() + feature_to_remove); //erase kth index of curr_features where
removing k yielded the highest accuracy.
        if(best_accuracy >= final_accuracy) {
            final_accuracy = best_accuracy;
```

```
                        final_features = curr_features;
                }else {
                        cout << endl << "(Warning, accuracy has decreased! Continuing search in case of local maxima)";
                }
                cout << endl << "Feature set {";
                for(int j = 0; j < curr_features.size(); ++j) {
                        if(j == curr_features.size() - 1) {
                                cout << curr_features.at(j);
                        }else {
                                cout << curr_features.at(j) << ",";
                        }
                }
                cout << "} was best, accuracy is " << setprecision(3) << best_accuracy * 100 << "%" << endl << endl;
        }
        cout << "Finished search! The best feature subset is {";
        for(int i = 0; i < final_features.size(); ++i){
                if(i == final_features.size() - 1) {
                        cout << final_features.at(i);
                }else{
                        cout << final_features.at(i) << ",";
                }
        }
        cout << "}, which has an accuracy of " << setprecision(3) << final_accuracy * 100 << "%" << endl << endl;
}

int main () {
        int input, numFeatures, numInstances;
        vector<vector<double>> data = readData();
        cout << endl << "Type the number of the algorithm you want to run." << endl << endl;
        cout << "1) Forward Selection" << endl << "2) Backward Elimination" << endl << endl;
        cin >> input;
        while(input!=1 && input!=2) {
                cout << "Invaid input, try again" << endl << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(),'\n');
                cin >> input;
        }
        numFeatures = data.at(0).size() - 1; //assuming each instance has same amount of features
        numInstances = data.size();
        cout << endl << "The dataset has " << numFeatures << " features (not including the class attribute), with " <<
numInstances << " instances." << endl << endl;
        cout << "Running nearest neighbor with all " << numFeatures << " features, using \"leaving-one-out\" evaluation, I get an
accuracy of ";
        vector<int> features;
        for(int i = 1; i < data.at(0).size() - 1; ++i) {
                features.push_back(i);
        }
        cout << setprecision(3) << crossValidate(data, features, features.size() + 1, 1) * 100 << "%" << endl << endl;
        if (input == 1) {
                forwardSelection(data);
        } else if(input == 2) {
                backwardElimination(data);
        }
        return 0;
}
```