Chi Chuan Howard Chen

SID: 862101733

Email: cchen277@ucr.edu

19 March 2021

Project 2 for CS170 Winter 2021, with Dr Eamonn Keogh.

All code is original, except:

1) “Get_time.h” was sourced from another class, CS142
2) I used <limits> to get my infinity for the cross validation.
3) I learned <fstream> and <sstream> from geeksforgeeks.org

In this project, we were tasked to perform Feature Selection with the Nearest Neighbor algorithm on a set of data. The objective was to find the set of features that most accurately determines a class of an object, using the Nearest Neighbor Algorithm. There are two methods that we used to identify this ideal set of features: forwards (adding each feature that increases the class identification accuracy), and backwards (deleting each feature to end up with a higher accuracy).

Figure 1 represents the results of running forward selection on my assigned small data set ("CS170_SMALLtestdata__6.txt").

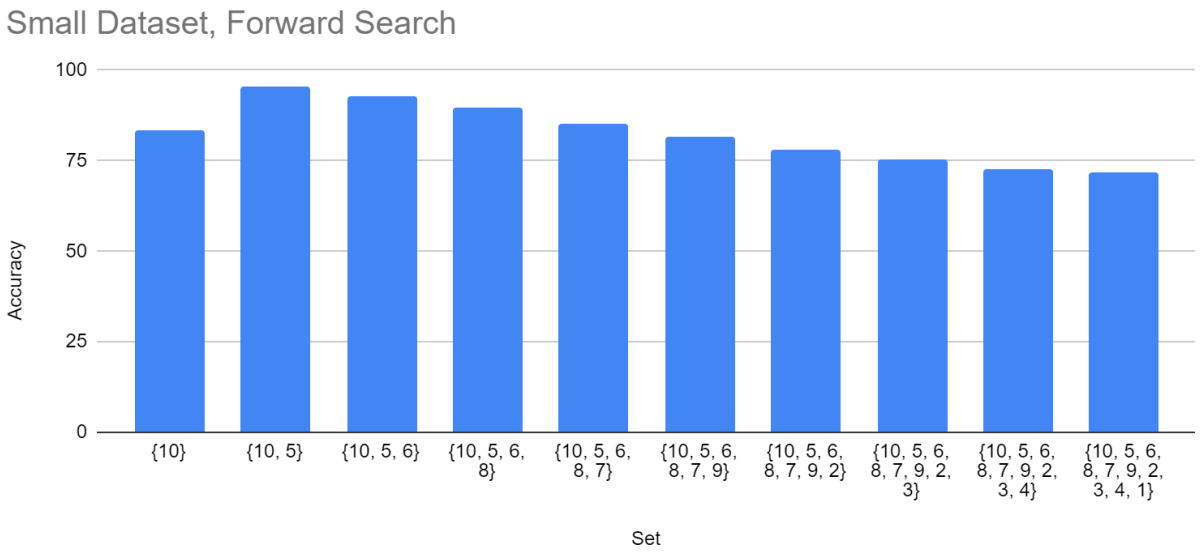

Figure 1: Accuracy of subsets of features discovered by forward selection

The search starts off with an empty set of features. At each depth of the search, one feature is added to the set to maximize the accuracy of the search at that depth until all sets are eventually added into the set of features. The subset of features that yields the highest accuracy in my test was {10, 5}, with an accuracy of 95.667%. Based on the results, it is evident that each added feature with the increasing depth was less correlated to the class identification than features 10

and 5, thus reducing the accuracy along the way. The final set with all features yields an accuracy of 71.667%.

Figure 2 depicts the results of running backward elimination on my assigned small data set ("CS170_SMALLtestdata__6.txt").

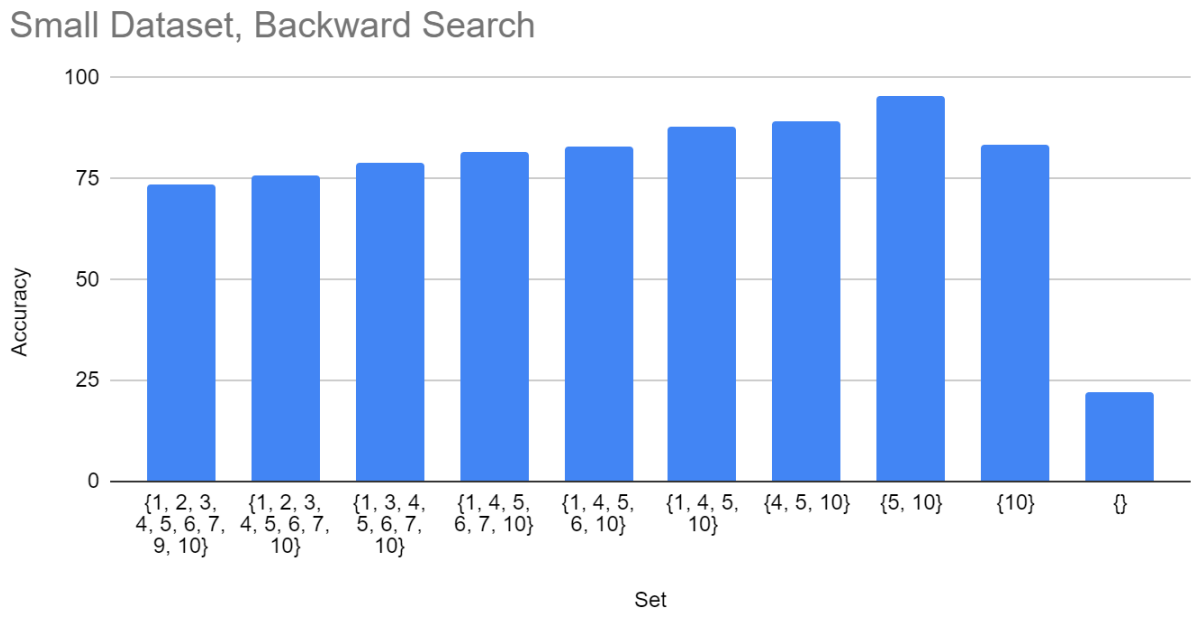

Figure 2: Accuracy of subsets of features discovered by backward elimination

This search starts off with the set of every feature available for the object. At each depth of the search, one feature is eliminated from the set to maximize the accuracy of the remaining subset. Although the elimination process and percentages don't completely mirror that of my forward search results, the most accurate subset discovered is still {10, 5} with an accuracy of 95.667%. The subset with the lowest accuracy is the empty set with an accuracy of 22%.

Conclusion for Small Dataset: Proven by both search algorithms, features '5' and '10' are the features that best define the class of the objects in my dataset. The inclusion of every other feature(s) only serves to weaken the correlation and accuracy of the feature set.

Conclusion for large dataSet: The most accurate subset of features is {55, 35} with an accuracy of 97.2% based on the forward search (shown in sheet 2 in Github).

Computational Effort for Search: I implemented my program and search algorithms using c++, and ran all experiments on my laptop with the Intel Core i7-9750H CPU and 32GB of RAM. The table below depicts the runtimes of the four searches I conducted.

	Small dataset	Large Dataset
Forward Selection	4.60449 seconds	395.5 seconds
Backward Elimination	5.25829 seconds	Unfinished (at level 28 as of 23:50)

Shown below is a single trace of my algorithm performing Forward Selection on the small dataset.

Welcome to the Feature Selection Algorithm.

Type in the name of the file to test : CS170_SMALLtestdata__6.txt

Type the number of the algorithm you want to run.

1) Forward Selection

2) Backward Elimination

Your algorithm of choice: 1

Begin Search

On level 1 of the search tree

Using features(s) {1} accuracy is 61.6667%

Using features(s) {2} accuracy is 67.3333%

Using features(s) {3} accuracy is 66%

Using features(s) {4} accuracy is 69.3333%

Using features(s) {5} accuracy is 69.6667%

Using features(s) {6} accuracy is 64.6667%

Using features(s) {7} accuracy is 64.6667%

Using features(s) {8} accuracy is 65%

Using features(s) {9} accuracy is 68.3333%

Using features(s) {10} accuracy is 83.3333%

Feature set {10} was best, accuracy is 83.3333%

On level 2 of the search tree

Using features(s) {10, 1} accuracy is 80.6667%

Using features(s) {10, 2} accuracy is 79%

Using features(s) {10, 3} accuracy is 81.3333%

Using features(s) {10, 4} accuracy is 81.6667%

Using features(s) {10, 5} accuracy is 95.6667%

Using features(s) {10, 6} accuracy is 80.6667%

Using features(s) {10, 7} accuracy is 81.3333%

Using features(s) {10, 8} accuracy is 77.6667%

Using features(s) {10, 9} accuracy is 81.3333%

Feature set {10, 5} was best, accuracy is 95.6667%

On level 3 of the search tree

Using features(s) {10, 5, 1} accuracy is 89%

Using features(s) {10, 5, 2} accuracy is 90.6667%

Using features(s) {10, 5, 3} accuracy is 91%

Using features(s) {10, 5, 4} accuracy is 89.3333%

Using features(s) {10, 5, 6} accuracy is 93%

Using features(s) {10, 5, 7} accuracy is 88%

Using features(s) {10, 5, 8} accuracy is 90.6667%

Using features(s) {10, 5, 9} accuracy is 92%

Feature set {10, 5, 6} was best, accuracy is 93%

On level 4 of the search tree

Using features(s) {10, 5, 6, 1} accuracy is 87%

Using features(s) {10, 5, 6, 2} accuracy is 84%

Using features(s) {10, 5, 6, 3} accuracy is 87.6667%

Using features(s) {10, 5, 6, 4} accuracy is 85%

Using features(s) {10, 5, 6, 7} accuracy is 85.3333%

Using features(s) {10, 5, 6, 8} accuracy is 89.6667%

Using features(s) {10, 5, 6, 9} accuracy is 88.3333%

Feature set {10, 5, 6, 8} was best, accuracy is 89.6667%

On level 5 of the search tree

Using features(s) {10, 5, 6, 8, 1} accuracy is 85%

Using features(s) {10, 5, 6, 8, 2} accuracy is 82%

Using features(s) {10, 5, 6, 8, 3} accuracy is 82.6667%

Using features(s) {10, 5, 6, 8, 4} accuracy is 80.3333%

Using features(s) {10, 5, 6, 8, 7} accuracy is 85.3333%

Using features(s) {10, 5, 6, 8, 9} accuracy is 83%

Feature set {10, 5, 6, 8, 7} was best, accuracy is 85.3333%

On level 6 of the search tree

Using features(s) {10, 5, 6, 8, 7, 1} accuracy is 78.6667%

Using features(s) {10, 5, 6, 8, 7, 2} accuracy is 78.6667%

Using features(s) {10, 5, 6, 8, 7, 3} accuracy is 78.3333%

Using features(s) {10, 5, 6, 8, 7, 4} accuracy is 77.3333%

Using features(s) {10, 5, 6, 8, 7, 9} accuracy is 81.6667%

Feature set {10, 5, 6, 8, 7, 9} was best, accuracy is 81.6667%

On level 7 of the search tree

Using features(s) {10, 5, 6, 8, 7, 9, 1} accuracy is 75.3333%

Using features(s) {10, 5, 6, 8, 7, 9, 2} accuracy is 78%

Using features(s) {10, 5, 6, 8, 7, 9, 3} accuracy is 77%

Using features(s) {10, 5, 6, 8, 7, 9, 4} accuracy is 76.6667%

Feature set {10, 5, 6, 8, 7, 9, 2} was best, accuracy is 78%

On level 8 of the search tree

Using features(s) {10, 5, 6, 8, 7, 9, 2, 1} accuracy is 73%

Using features(s) {10, 5, 6, 8, 7, 9, 2, 3} accuracy is 75.3333%

Using features(s) {10, 5, 6, 8, 7, 9, 2, 4} accuracy is 74.6667%

Feature set {10, 5, 6, 8, 7, 9, 2, 3} was best, accuracy is 75.3333%

On level 9 of the search tree

Using features(s) {10, 5, 6, 8, 7, 9, 2, 3, 1} accuracy is 69.6667%

Using features(s) {10, 5, 6, 8, 7, 9, 2, 3, 4} accuracy is 72.6667%

Feature set {10, 5, 6, 8, 7, 9, 2, 3, 4} was best, accuracy is 72.6667%

On level 10 of the search tree

Using features(s) {10, 5, 6, 8, 7, 9, 2, 3, 4, 1} accuracy is 71.6667%

Feature set {10, 5, 6, 8, 7, 9, 2, 3, 4, 1} was best, accuracy is 71.6667%

Finished Search! The best feature subset so is {10, 5} with an accuracy of 95.6667%

Code: Below is my code for this project (Github link here):

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <limits>
#include <cmath>
#include "get_time.h"
using namespace std;

void printFeatureSet(vector<int> &v) {
    cout << "{";
    for (int i = 0; i < v.size(); i++) {
        if (i+1 < v.size()) {
            cout << v.at(i) << ", ";
        }
        else {
            cout << v.at(i);
        }
    }
    cout << "}";
}

void remove_k(vector<int> &v, int k) {
    vector<int>::iterator it = find(v.begin(), v.end(), k);
    v.erase(it);
    // printFeatureSet(v);
    // cout << endl;
}

double findDistance(vector<double> &object_to_classify, vector<double>
&neighbor_to_classify, vector<int> &updated_feature_set) {
    double sum = 0;
    double distance = 0;
    //calculate sum
```

```
    for (int i = 1; i < object_to_classify.size(); i++) {
        // account for feature i if i is found in updated_feature_set
        if (find(updated_feature_set.begin(), updated_feature_set.end(),
i) != updated_feature_set.end()) {
            // cout << i << ", ";
            double difference = neighbor_to_classify.at(i) -
object_to_classify.at(i);
            sum += pow(difference, 2);
        }
    }
    // cout << endl;
    distance = sqrt(sum);
    return distance;
}

double leave_one_out_cross_validation(vector<vector<double>> &data,
vector<int> &set_of_features, int k, bool isForward) {
    int number_correctly_classfied = 0; // keep track of correct
classifications
    // create new set to include k
    vector<int> updated_feature_set = set_of_features;
    if (isForward) {
        updated_feature_set.push_back(k);
    }
    else {
        remove_k(updated_feature_set, k);
    }
    // print updated_feature_set
    // cout << "       Using features(s) ";
    // printFeatureSet(updated_feature_set);

    //iterate through each object
    for (int i = 0; i < data.size(); i++) {
        vector<double> object_to_classify = data.at(i); // current object
        double label_object_to_classify = object_to_classify.at(0); //
object class
        int nearest_neighbor = 0; // keep track of nearest neighbor
```

```
        double label_neighbor_to_classify = 0; // keep track of nearest
neighbor's class
        double inf = numeric_limits<double>::infinity();
        double nearest_neighbor_distance = inf; // keep track of nearest
neighbor's distance

        //compare object with each of its neighbors
        for (int n = 0; n < data.size(); n++) {
            if (n != i) {
                // cout << "--Ask if " << i+1 << " is the nearest neighbor
with " << n+1 << endl;
                vector<double> neighbor_to_classify = data.at(n);

                double distance = findDistance(object_to_classify,
neighbor_to_classify, updated_feature_set); // get distance between object
and neighbor

                //update nearest neighbor, its distance, and class
                if (distance < nearest_neighbor_distance) {
                    nearest_neighbor_distance = distance;
                    nearest_neighbor = n+1;
                    label_neighbor_to_classify =
neighbor_to_classify.at(0);
                }
            }
        }

        // update number of correct answers
        if (label_object_to_classify == label_neighbor_to_classify) {
            number_correctly_classfied++;
        }

    }
    // print accuracy
    double accuracy = double(number_correctly_classfied) /
double(data.size());
    // cout << " accuracy is " << accuracy*100 << "%" << endl;
```

```
    return accuracy;
}

void forwardFeatureSearch(vector<vector<double>> &data) {
    vector<int> current_set_of_features; // start with an empty set of
features
    vector<int> best_feature_subset; // keep track of most accurate
feature set
    double overall_best_accuracy = 0.0;
    cout << "Begin Search" << endl;

    //iterate through each item
    for (int i = 1; i < data.at(0).size(); i++) {
        cout << "On level " << i << " of the search tree" << endl;
        int feature_to_add_at_this_level = 0; // keep track of feature to
add
        double level_best_accuracy = 0.0; // keep track of best accuracy

        // iterate through features
        for (int k = 1; k < data.at(0).size(); k++) {
            // check accuracy of curr set + k if k doesn't already exist
in current featureset
            if (find(current_set_of_features.begin(),
current_set_of_features.end(), k) == current_set_of_features.end()) {
                // get accuracy after adding k to current set
                double accuracy = leave_one_out_cross_validation(data,
current_set_of_features, k, 1);
                // update highest accuracy (from feature k) if applicable
                if (accuracy > level_best_accuracy) {
                    level_best_accuracy = accuracy;
                    feature_to_add_at_this_level = k;
                }
            }
        }
```

```
        current_set_of_features.push_back(feature_to_add_at_this_level);
// update curr feature set
        if (level_best_accuracy > overall_best_accuracy) { // update best
feature set and accuracy if current set is better
            overall_best_accuracy = level_best_accuracy;
            best_feature_subset = current_set_of_features;
        }

        cout << "Feature set ";
        printFeatureSet(current_set_of_features);
        cout << " was best, accuracy is " << level_best_accuracy*100 <<
"%" << endl;
    }
    cout << "\nFinished Search! The best feature subset so is ";
    printFeatureSet(best_feature_subset);
    cout << " with an accuracy of " << overall_best_accuracy*100 << "%" <<
endl;
}

void backwardFeatureSearch(vector<vector<double>> &data) {
    vector<int> current_set_of_features; // start with an empty set of
features
    for (int h = 1; h < data.at(0).size(); h++) {
        current_set_of_features.push_back(h);
    }
    // printFeatureSet(current_set_of_features);
    vector<int> best_feature_subset; // keep track of most accurate
feature set
    double overall_best_accuracy = 0.0;
    cout << "Begin Search" << endl;

    //iterate through each item
    for (int i = 1; i < data.at(0).size(); i++) {
        cout << "On level " << i << " of the search tree" << endl;
        int feature_to_remove_at_this_level = 0; // keep track of feature
to remove
        double level_best_accuracy = 0.0; // keep track of best accuracy
```

```
        // iterate through features
        for (int k = 1; k < data.at(0).size(); k++) {
            // check accuracy of curr set without k if k hasn't been
removed from current featureset
            if (find(current_set_of_features.begin(),
current_set_of_features.end(), k) != current_set_of_features.end()) {
                // get accuracy after removing k from current set
                double accuracy = leave_one_out_cross_validation(data,
current_set_of_features, k, 0);
                // update highest accuracy (from feature k) if applicable
                if (accuracy > level_best_accuracy) {
                    level_best_accuracy = accuracy;
                    feature_to_remove_at_this_level = k;
                }
            }
        }

        remove_k(current_set_of_features,
feature_to_remove_at_this_level); // update curr feature set
        if (level_best_accuracy > overall_best_accuracy) { // update best
feature set and accuracy if current set is better
            overall_best_accuracy = level_best_accuracy;
            best_feature_subset = current_set_of_features;
        }

        cout << "Feature set ";
        printFeatureSet(current_set_of_features);
        cout << " was best, accuracy is " << level_best_accuracy*100 <<
"%" << endl;
    }
    cout << "\nFinished Search!The best feature subset so is ";
    printFeatureSet(best_feature_subset);
    cout << " with an accuracy of " << overall_best_accuracy*100 << "%" <<
endl;
}
```

```
int main() {
    string fileName;
    int algChoice;

    cout << "Welcome to the Feature Selection Algorithm." << endl;
    cout << "Type in the name of the file to test : ";
        cin >> fileName;
    cout << "Type the number of the algorithm you want to run." << endl;
        cout << "    1) Forward Selection" << endl;
        cout << "    2) Backward Elimination" << endl;
        cout << "    Your algorithm of choice: ";
        cin >> algChoice;
        cout << endl;

    // save input data into 2d vector 'data'
    vector<vector<double>> data;
    // ifstream fin("CS170_SMALLtestdata__6.txt");
    ifstream fin(fileName);
    if (fin.is_open()) {
        double element;
        while (fin.good()) {
            vector<double> temp;
            double element;
            string input;
            getline(fin, input);
            stringstream sstream;
            sstream << input;
            while (sstream >> element) {
                temp.push_back(element);
            }
            data.push_back(temp);
        }
        data.pop_back();
    }
    else {
        cout << "unable to open input file" << endl;
        return 1;
```

```
    }
    fin.close();

    if (algChoice == 1) {
        timer s;
        forwardFeatureSearch(data);
        s.stop();
        cout << "Time elapsed: " << s.get_total() << endl;
        cout << endl;
    }
    else if (algChoice == 2) {
        timer t;
        backwardFeatureSearch(data);
        t.stop();
        cout << "Time elapsed: " << t.get_total() << endl;
    }
    else {
        cout << "Algorithm Choice doesn't exist" << endl;
        return 1;
    }

    return 0;
}
```