

Thomas Shiels  
ID 861151979  
[tshie001@ucr.edu](mailto:tshie001@ucr.edu)  
17-December-2018

Instructor: Dr. Eamonn Keogh

In completing this homework, I consulted...

- <https://stackoverflow.com/questions/25188288/how-to-calculate-distance-between-points-using-vectors> (for clarification on calculating the Euclidian distance in C++)
- MachineLearning001 Lecture notes from Class
- [https://www.researchgate.net/post/What\\_is\\_leave-one-out\\_cross\\_validation\\_How\\_can\\_I\\_use\\_it\\_for\\_image\\_fusion](https://www.researchgate.net/post/What_is_leave-one-out_cross_validation_How_can_I_use_it_for_image_fusion) (for clarification on leave-one-out cross validation)

All the important code is original. Unimportant subroutines that are not completely original are...

- `<fstream>` from the C++ standard library for reading in data
- `<vector>` from the C++ standard library for holding arrays
- `<cmath>` for calculations relating to Euclidian distance
- `<time.h>` for measuring elapsed time of functions

# Introduction

In this project we were asked to perform feature selection with the nearest neighbor algorithm and leave-one-out cross validation. Tests were performed on two data sets with varying numbers of features. Each example had two possible classes, which were determined by its features. The object of this assignment was to use forward selection and backward elimination search algorithms to determine the best features for classification on the nearest neighbor algorithm.

## Nearest Neighbor

Nearest neighbor is a supervised classification algorithm that takes in a dataset with  $n$  number of features and maps them in  $n$ -dimensional space. It determines the class of an example by calculating its distance from its closest neighbor. The algorithm uses a nested for loop to separate the data into the training set and the testing set. In the  $k$ -nearest-neighbors version of the algorithm, the class is determined by the nearest  $k$  number of neighbors, whereas in leave-one-out cross validation, each example is tested against every other example. The number of correct classifications divided by the number of examples determines the accuracy of the algorithm.

## Forward Selection

Greedy forward selection is a search algorithm which consists of a nested for loop that cycles through the best combination of features, starting with one feature. When the first best feature is determined, it combines it with every other combination of two features, and so on, until the final level of the search tree. The algorithm keeps track of the best combination of features, so the final result may not contain most or all of the features.

## Backward Elimination

Backward elimination is a complement to forward selection in that it starts out with every feature, and progressively removes irrelevant features. This algorithm is also greedy, meaning that at each step, the best immediate result is chosen without taking into account possible future results.

## Custom Algorithm

My idea for the custom algorithm was to reduce the time it takes to finish at a minimal cost to accuracy. Since nearest neighbor is highly susceptible to irrelevant features, I thought it would be useful to prematurely “prune” the dataset before attempting forward selection. I wanted to find a small proportion of the most accurate individual features, and then run forward selection on just those features. For the purposes of this assignment, I kept the proportion constant at the top three features, but would ideally use a ratio of the total number of features. Due to the arrangement of my algorithm, the dataset was reformatted to contain the top three features in a

different order to the original. So while I was easily able to recover which features remained on the small dataset due to its small size, the exact features for the large dataset were harder to recover.

## Results

### Small Data Set

Using CS170\_SMALLtestdata\_\_59.txt

- **Forward selection:** [5, 10] with an accuracy of 97% at 2.02 seconds
- **Backward elimination:** [5, 10] with an accuracy of 97% at 2.32 seconds

### Large Data Set

Using CS170\_LARGEtestdata\_\_107.txt

- **Forward selection:** [22,6] with an accuracy of 96.5% at 9 minutes 15 seconds
- **Backward elimination:** [5, 6, 12, 14, 18, 20, 23, 27, 28, 31, 34, 40, 42, 43, 44, 46, 49, 51, 55, 58, 63, 66, 67, 71, 72, 74, 76, 78, 80, 81, 83, 84, 85, 86, 90, 91, 92, 94, 95, 96, 97, 98, 99, 100] with an accuracy of 86% at 15 minutes 24 seconds.

### Custom Algorithm

- **Large dataset:** [22] with an accuracy of 85% in 3.4 seconds.
- **Small dataset:** [5,10] with an accuracy of 97% in 0.49 seconds.

## Conclusion

The forward selection and backward elimination provided the same results in the small dataset, but backward elimination performed worse in the large dataset. Forward selection also seemed to be consistently faster than backward elimination in both instances. Overall, forward selection performed better. The custom algorithm also provided the same results as forward selection on the small dataset, and returned one strong feature in the large dataset. The time it took to complete on both data sets was noticeably faster, with similar accuracies.

# Trace

```
tshie001@cs_private:~/workspace/cs170/project2 $ ./a.out
Welcome to Bootie Wooster's Feature Selection Algorithm.
Type in the name of the file to test: CS170_SMALLtestdata__59.txt

Type the number of the algorithm you want to run.
1. Forward Selection
2. Backward Elimination
3. Custom algorithm
1
On level 1 of the search tree
Using feature(s) {1} accuracy is 71.5%
Using feature(s) {2} accuracy is 71.5%
Using feature(s) {3} accuracy is 72%
Using feature(s) {4} accuracy is 69%
Using feature(s) {5} accuracy is 79.5%
Using feature(s) {6} accuracy is 69%
Using feature(s) {7} accuracy is 72%
Using feature(s) {8} accuracy is 72%
Using feature(s) {9} accuracy is 66%
Using feature(s) {10} accuracy is 81.5%
On level 2 of the search tree
Using feature(s) {10,1} accuracy is 84%
Using feature(s) {10,2} accuracy is 83.5%
Using feature(s) {10,3} accuracy is 83.5%
Using feature(s) {10,4} accuracy is 83%
Using feature(s) {10,5} accuracy is 97%
Using feature(s) {10,6} accuracy is 80%
Using feature(s) {10,7} accuracy is 86%
Using feature(s) {10,8} accuracy is 81.5%
Using feature(s) {10,9} accuracy is 80.5%
On level 3 of the search tree
Using feature(s) {10,5,1} accuracy is 95%
Using feature(s) {10,5,2} accuracy is 89.5%
Using feature(s) {10,5,3} accuracy is 91.5%
Using feature(s) {10,5,4} accuracy is 91%
Using feature(s) {10,5,6} accuracy is 91%
Using feature(s) {10,5,7} accuracy is 91%

On level 6 of the search tree
Using feature(s) {10,5,1,4,3,2} accuracy is 77.5%
Using feature(s) {10,5,1,4,3,6} accuracy is 77.5%
Using feature(s) {10,5,1,4,3,7} accuracy is 79.5%
Using feature(s) {10,5,1,4,3,8} accuracy is 76%
Using feature(s) {10,5,1,4,3,9} accuracy is 83%
(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {10,5,1,4,3,9,9} was best, accuracy is 83%

On level 7 of the search tree
Using feature(s) {10,5,1,4,3,9,2} accuracy is 79.5%
Using feature(s) {10,5,1,4,3,9,6} accuracy is 74%
Using feature(s) {10,5,1,4,3,9,7} accuracy is 76.5%
Using feature(s) {10,5,1,4,3,9,8} accuracy is 75.5%
(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {10,5,1,4,3,9,2,2} was best, accuracy is 79.5%

On level 8 of the search tree
Using feature(s) {10,5,1,4,3,9,2,6} accuracy is 74.5%
Using feature(s) {10,5,1,4,3,9,2,7} accuracy is 77.5%
Using feature(s) {10,5,1,4,3,9,2,8} accuracy is 78.5%
(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {10,5,1,4,3,9,2,8,8} was best, accuracy is 78.5%

On level 9 of the search tree
Using feature(s) {10,5,1,4,3,9,2,8,6} accuracy is 69%
Using feature(s) {10,5,1,4,3,9,2,8,7} accuracy is 76%
(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {10,5,1,4,3,9,2,8,7,7} was best, accuracy is 76%

On level 10 of the search tree
Using feature(s) {10,5,1,4,3,9,2,8,7,6} accuracy is 69%
(Warning, accuracy has decreased! Continuing search in case of local maxima)
Feature set {10,5,1,4,3,9,2,8,7,6,6} was best, accuracy is 69%

Finished search! The best feature subset is {10,5}, which has an accuracy of 97%
tshie001@cs_private:~/workspace/cs170/project2 $ █
```

# Source Code

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <time.h>

using namespace std;

double e_distance(vector<double> x, vector<double> y, vector<int> curr_set, int j,
    bool fwd);

double loo_nn(vector< vector<double> > data, vector<int> curr_set, int j,
    bool fwd);

void forward_selection(vector< vector<double> > data);
void backward_elimination(vector< vector<double> > data);

void custom_alg(vector< vector<double> > data)
{
    vector<int> top_three(3);
    vector<double> accuracies;
    vector<int> current_set_of_features;
    current_set_of_features.resize(1);
    double max_acc = 0;

    //test all features individually
    for (int i = 1; i < data.at(0).size(); ++i)
    {
        current_set_of_features.at(0) = i;
        double temp_acc = loo_nn(data, current_set_of_features, 0, false);
        accuracies.push_back(temp_acc);
    }

    //capture top three features
    for (int i = 0; i < accuracies.size(); ++i)
    {
        if (accuracies.at(i) > max_acc)
        {
            max_acc = accuracies.at(i);
            top_three.at(0) = i + 1;
        }
    }
    max_acc = accuracies.at(0);
    for (int i = 0; i < accuracies.size(); ++i)
    {
        if ((accuracies.at(i) > max_acc) && (i + 1 != top_three.at(0)))
        {
            max_acc = accuracies.at(i);
            top_three.at(1) = i + 1;
        }
    }
    max_acc = accuracies.at(0);
    for (int i = 0; i < accuracies.size(); ++i)
    {
        if ((accuracies.at(i) > max_acc) && (i + 1 != top_three.at(1)) &&
            (i + 1 != top_three.at(0)))
        {
            max_acc = accuracies.at(i);
            top_three.at(2) = i + 1;
        }
    }

    //create new data matrix with only top three features
    vector< vector<double> > data2(data.size());
    vector<double> temp_data(4);
    for (int i = 0; i < data.size(); ++i)
    {
        temp_data.at(0) = data.at(i).at(0);
        temp_data.at(1) = data.at(i).at(top_three.at(0));
        temp_data.at(2) = data.at(i).at(top_three.at(1));
        temp_data.at(3) = data.at(i).at(top_three.at(2));

        data2.at(i) = temp_data;
    }

    forward_selection(data2);
    return;
}

//check to see if two features match
```

26:37 C and C++ Spaces: 4

```

//check to see if two features match
bool intersect(vector<int> curr_feats, int index)
{
    for (int i = 0; i < curr_feats.size(); ++i)
    {
        if (curr_feats.at(i) == index)
        {
            return true;
        }
    }
    return false;
}

//used to remove a feature during backward elimination
vector<int> remove_feat(vector<int> feat, int index)
{
    for (int i = 0; i < feat.size(); ++i)
    {
        if (feat.at(i) == index)
        {
            feat.erase(feat.begin() + i);
            return feat;
        }
    }
    return feat;
}

void backward_elimination(vector< vector<double> > data)
{
    vector<int> current_set_of_features;
    vector<int> best_feats;
    double acc_total = 0;

    //fill set with all features
    for (int i = 1; i < data.at(0).size(); ++i)
    {
        current_set_of_features.push_back(i);
    }
}

```

26:37 C and C++ Spaces: 4

```

//1.o.o nearest neighbor starting with n - 1 features, and
//check best accuracy
for (int i = 1; i < data.at(0).size() - 1; ++i)
{
    int feat_to_remove;
    double best_acc_so_far = 0;

    for (int j = 1; j < data.at(0).size(); ++j)
    {
        if (intersect(current_set_of_features, j))
        {
            vector<int> temp = remove_feat(current_set_of_features, j);
            cout << "Using feature(s) {"<
            for (int k = 0; k < temp.size() - 1; ++k)
            {
                cout << temp.at(k) << ","<
            }
            cout << temp.at(temp.size() - 1) << "} accuracy is "<

            double acc = loo_nn(data, temp, j, false);
            cout << acc << endl;
            if (acc > best_acc_so_far)
            {
                best_acc_so_far = acc;
                feat_to_remove = j;
            }
        }
    }
    current_set_of_features = remove_feat(current_set_of_features,
        feat_to_remove);

    if (best_acc_so_far > acc_total)
    {
        acc_total = best_acc_so_far;
        best_feats = current_set_of_features;
    }
    else
    {
        cout << "(Warning, accuracy has decreased!";
        cout << "case of local maxima)" << endl;
    }
}

```

26:37 C and C++ Spaces: 4

```

        cout << "case of local maxima" << endl;
    }
    cout << "Feature set {";
    for (int k = 0; k < current_set_of_features.size() - 1; ++k)
    {
        cout << current_set_of_features.at(k) << ",";
    }
    cout << current_set_of_features.at(current_set_of_features.size() - 1);
    cout << "} was best, accuracy is ";
    cout << best_acc_so_far * 100 << "%" << endl;
    cout << endl;
}

cout << "Finished search! The best feature subset is {";
for (int k = 0; k < best_feats.size() - 1; ++k)
{
    cout << best_feats.at(k) << ",";
}
cout << best_feats.at(best_feats.size() - 1);
cout << "} which has an accuracy of " << acc_total * 100 << "%" << endl;

return;
}

void forward_selection(vector< vector<double> > data)
{
    vector<int> current_set_of_features;
    vector<int> best_feats;
    double acc_total = 0;

    //for every feature
    for (int i = 1; i < data.at(0).size(); ++i)
    {
        cout << "On level " << i << " of the search tree" << endl;
        int feat_to_add_this_level = 0;
        double best_acc_so_far = 0;

        //1.o.o nearest neighbor
        for (int j = 1; j < data.at(0).size(); ++j)

```

26:37 C and C++ Spaces: 4

```

        //do not compare a feature to itself
        if (!intersect(current_set_of_features, j))
        {
            cout << "Using feature(s) {";
            for (int k = 0; k < current_set_of_features.size(); ++k)
            {
                cout << current_set_of_features.at(k) << ",";
            }
            cout << j << "} accuracy is ";

            double acc = loo_nn(data, current_set_of_features, j, true);

            cout << acc * 100 << "%" << endl;
            if (acc > best_acc_so_far)
            {
                best_acc_so_far = acc;
                feat_to_add_this_level = j;
            }
        }
    }
    //check total accuracy
    current_set_of_features.push_back(feat_to_add_this_level);
    if (best_acc_so_far > acc_total)
    {
        acc_total = best_acc_so_far;
        best_feats = current_set_of_features;
    }
    else
    {
        cout << "(Warning, accuracy has decreased! Continuing search in case ";
        cout << "of local maxima)" << endl;
        cout << "Feature set {";
        for (int n = 0; n < current_set_of_features.size(); ++n)
        {
            cout << current_set_of_features.at(n) << ",";
        }
        cout << current_set_of_features.at(current_set_of_features.size() - 1);
        cout << "} was best, accuracy is ";
        cout << best_acc_so_far * 100 << "%" << endl;
    }
}

```

26:37 C and C++ Spaces: 4

```

    }

    cout << "Finished search! The best feature subset is {"";
    for (int n = 0; n < best_feats.size() - 1; ++n)
    {
        cout << best_feats.at(n) << ",";
    }
    cout << best_feats.at(best_feats.size() - 1);
    cout << "}, which has an accuracy of " << acc_total * 100 << "%" << endl;


    return;
}

double loo_nn(vector< vector<double> > data, vector<int> curr_set, int index,
    bool fwd)
{
    double num_correct = 0;

    //for every example
    for (int i = 0; i < data.size(); ++i)
    {
        vector<double> test = data.at(i);
        double min_dist = 999;
        vector<double> min_n;

        //find distance from every other example
        for (int j = 0; j < data.size(); ++j)
        {
            if (j != i)
            {
                double dist = e_distance(test, data.at(j), curr_set, index, fwd);
                if (dist < min_dist)
                {
                    min_dist = dist;
                    min_n = data.at(j);
                }
            }
        }
    }
}

```

26:37 C and C++ Spaces: 4 

```

    }

    //sum number of correct guesses
    if ((int)min_n.at(0) == (int)test.at(0))
    {
        ++num_correct;
    }
}

double tot_acc = num_correct / (double)data.size();


return tot_acc;
}

double e_distance(vector<double> x, vector<double> y, vector<int> curr_set, int index,
    bool fwd)
{
    double dist = 0;
    for (int i = 0; i < curr_set.size(); ++i)
    {
        dist += pow(x.at(curr_set.at(i)) - y.at(curr_set.at(i)), 2);
    }
    if (fwd)
    {
        dist += pow((x.at(index) - y.at(index)), 2);
    }

    dist = sqrt(dist);
    return dist;
}

int main()
{
    ifstream fin;
    double input = 0;
    vector< vector<double> > data;
    cout << "Welcome to Bootie Wooster's Feature Selection Algorithm." << endl;
    cout << "Type in the name of the file to test: ";
}

```

26:37 C and C++ Spaces: 4 



```

int main()
{
    ifstream fin;
    double input = 0;
    vector< vector<double> > data;
    cout << "Welcome to Bootie Wooster's Feature Selection Algorithm." << endl;
    cout << "Type in the name of the file to test: ";

    string test_file;
    cin >> test_file;
    cout << endl;
    if (test_file == "CS170_SMALLtestdata__59.txt")
    {
        fin.open("CS170_SMALLtestdata__59.txt");
    }
    else if (test_file == "CS170_LARGEtestdata__107.txt")
    {
        fin.open("CS170_LARGEtestdata__107.txt");
    }
    else
    {
        cout << "File not recognized.";
        return 0;
    }
    if (!fin.is_open())
    {
        cout << "Error opening." << endl;
        return 0;
    }
    else
    {
        for (int i = 0; i < 200; ++i)
        {
            vector<double> temp;

            if (test_file == "CS170_LARGEtestdata__107.txt")
            {
                for (int j = 0; j < 101; ++j)
            }

            // input,
            temp.push_back(input);

        }

        data.push_back(temp);
    }
    fin.close();
}

cout << "Type the number of the algorithm you want to run." << endl;
cout << "1. Forward Selection" << endl << "2. Backward Elimination"
<< endl << "3. Custom algorithm" << endl;
int alg;
cin >> alg;

if (alg == 1)
{
    forward_selection(data);
}
else if (alg == 2)
{
    backward_elimination(data);
}
else
{
    //Custom Algorithm
    //Since nn is highly susceptible to irrelevant features
    //likely that most are irrelevant
    //test all features individually, and retain top three
    //"quick and dirty" approach that should improve speed
    //at minimal cost to accuracy

    custom_alg(data);
}

return 0;
}

```