Yuwei Zhang
SID# X719520 (concurrent enrollment, from the extension UCP program)
Email: yzhan995@ucr.edu
June 10, 2023

Project 2 for CS 205 Spring 2023, with Dr Eamonn Keogh.

When completing this assignment, I referred to the following materials:
Machine Learning slides and project_2_briefing slides
C++User Manual (cppreference. com)

My implementation method for this assignment is C++, and all the code is original

Introduction

Given a dataset, each instance has K features. Select a subset of the feature set. It is best to classify the dataset according to this subset. This assignment uses two methods (forward selection and backward elimination) to implement Feature Selection and verifies their effectiveness on two datasets of different sizes, using the language c++. Full code available at github.com/yzhan995/cs205_proj2

Method Introduction

Forward selection

Forward selection first initializes the set S to be empty, and each time it greedily selects the best feature from all features that are not in S to join the set (evaluating the feature performance by calculating the accuracy of S). This search method can obtain a path with a length of K (where K is the number of features), and the set with the best results on this path is the output from the forward selection algorithm.

Backward elimination

Contrary to Forward selection, a backward elimination initializes S as {1,2,..., K}, greedily extracting a feature from S each time. Similarly, this search method can also obtain a path with a length of K, and the set with the best results on this path is the output from the backward elimination algorithm.

Implementation

I have selected two samples of different sizes: :

- CS170_small_Data__1.txt (1000instances, 10features)
- CS170_large_Data__1.txt (2000instances, 20features)

Then I implemented the forward selection and backward elimination on the two samples.

Small instance

The experimental results of the two methods are shown in Figure 1 and Figure 2:
For Forward selection, the search starts from an empty set and greedily selects feature 8, achieving an accuracy of 85.0%. Then feature 2 is added, improving the accuracy to 96.6%. The subsequent increase in features leads to a gradual decrease in accuracy, ultimately resulting

in a set that includes all features, with an accuracy of 78.0%.

For the Backward Elimination, the search starts with a set containing all the features, with an accuracy of 78.0% (the same as the end state of the forward selection). The greedy deletion of feature 9 results in an accuracy of 80.7%, followed by the subsequent deletion of features 5, 7, 3, 4, 10, 6, and 1, resulting in the set {2, 8} and achieving the highest accuracy of 96.6%. Finally, feature 8 is deleted, reducing the accuracy to 85.0%

Conclusion: In this experiment, both search methods found the same optimal solution {2,8}, and the accuracy was significantly improved compared to the neighboring solutions of {2,8} (such as {2}, {8}, etc.). Therefore, I believe that the best feature set on this dataset is {2,8}, with the accuracy of 96.6%.

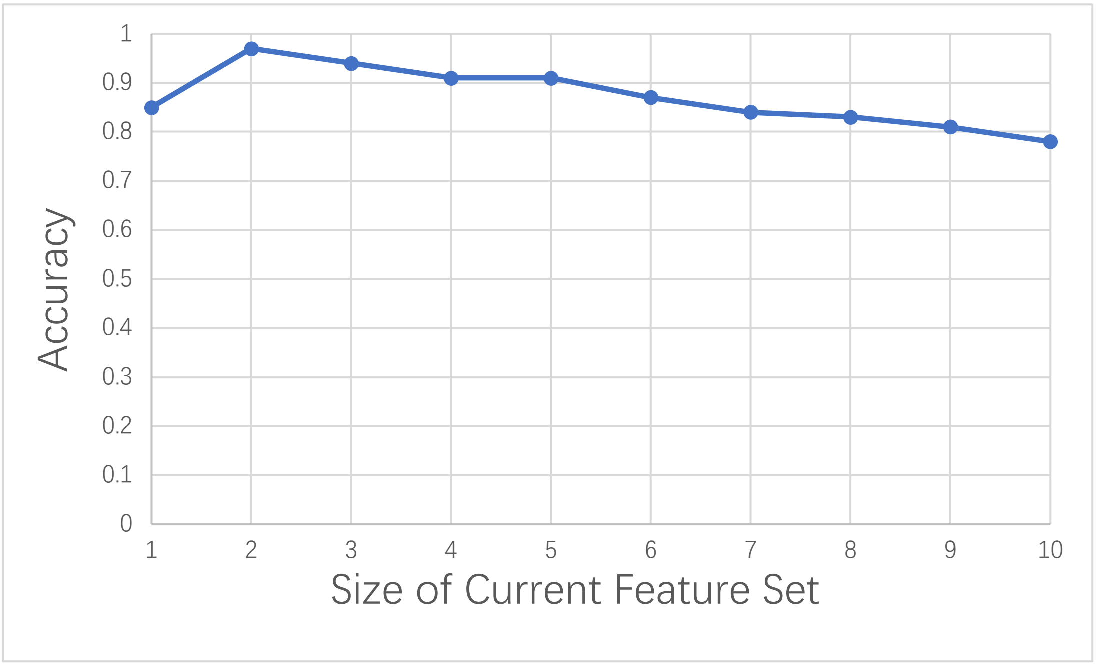

Figure 1 Forward Selection on CS170_small_Data__1

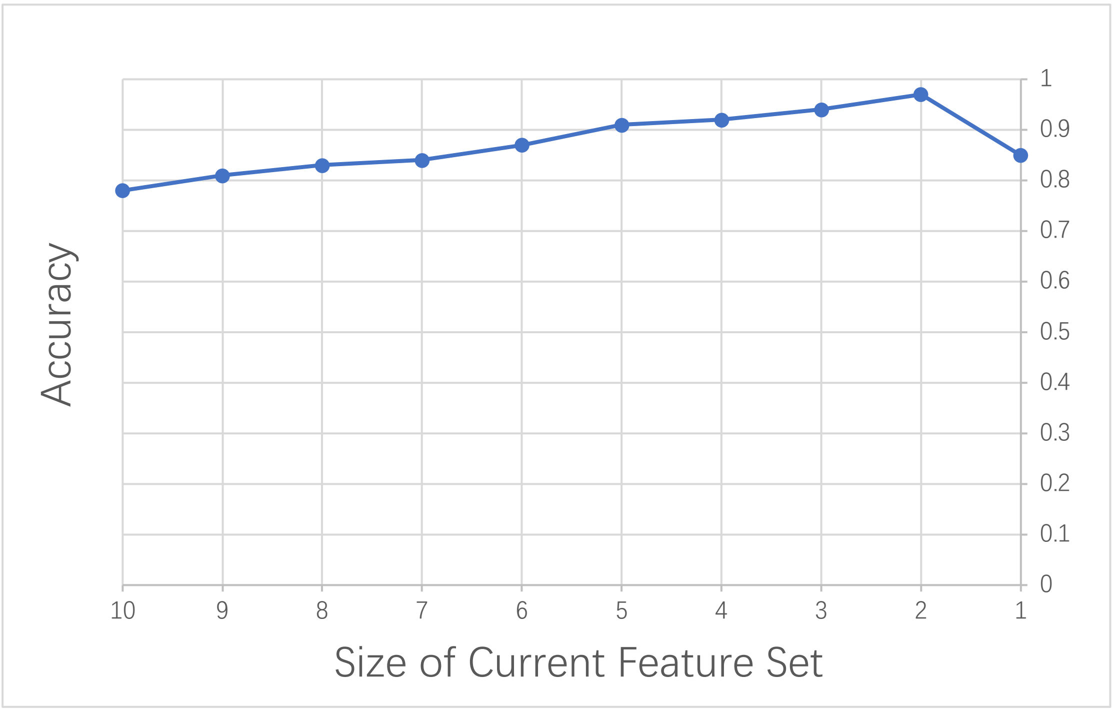

Figure 2 Backward Elimination on CS170_small_Data__1

Large instance

The results of the two methods on the large dataset are shown in Figure 3 and Figure 4:
For Forward selection, the search starts from an empty set and greedily selects feature 11, achieving an accuracy of 85.0%. Then feature 17 is added, improving the accuracy to 97.0%. The subsequent increase in features leads to a gradual decrease in accuracy, ultimately resulting in a set that includes all features, with an accuracy of 71.1%.

For the Backward Elimination, the search starts with a set containing all the features, with an accuracy of 71.1% (the same as the end state of the forward selection). The greedy deletion of feature 9 results in an accuracy of 73.2%, followed by the deletion of features 13, 14, 20,..., resulting in the set {11,17} and reaching the highest accuracy of 97.0%. Finally, feature 17 is deleted, reducing the accuracy to 85.0%

Conclusion: In this example, both search methods found the same optimal solution {11,17}, and the accuracy was significantly improved compared to the neighboring solutions of {11,17} (such as {11}, {17}, etc.). Therefore, I believe that the best feature set on this dataset is {11,17}, corresponding to an accuracy of 97.0%

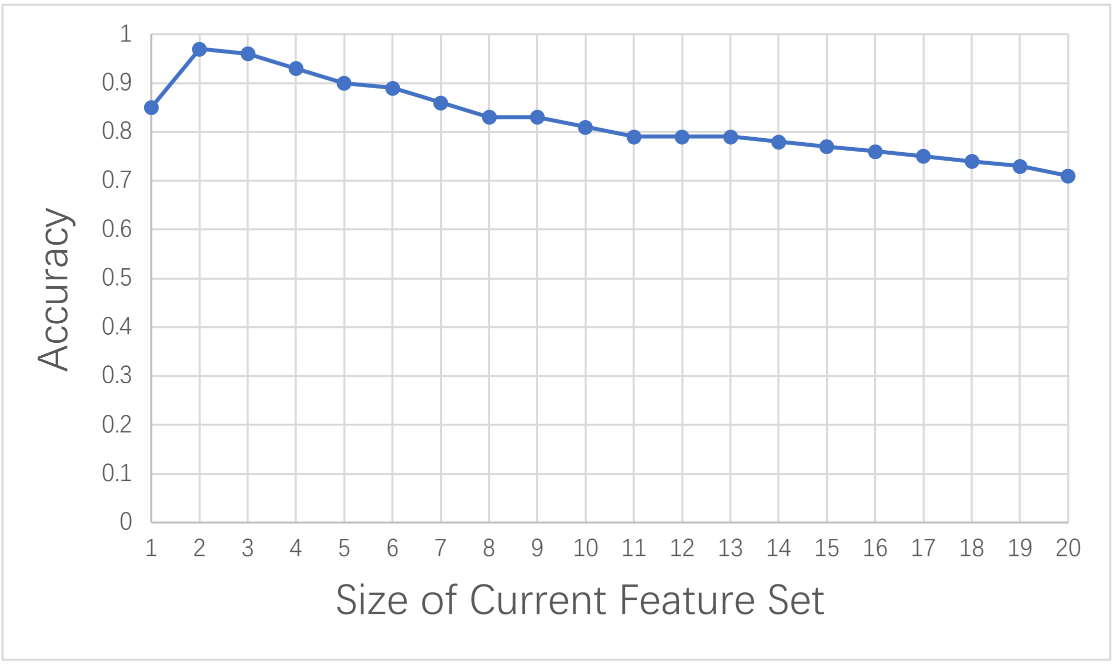

Figure 3 Forward Selection on CS170_large_Data__1

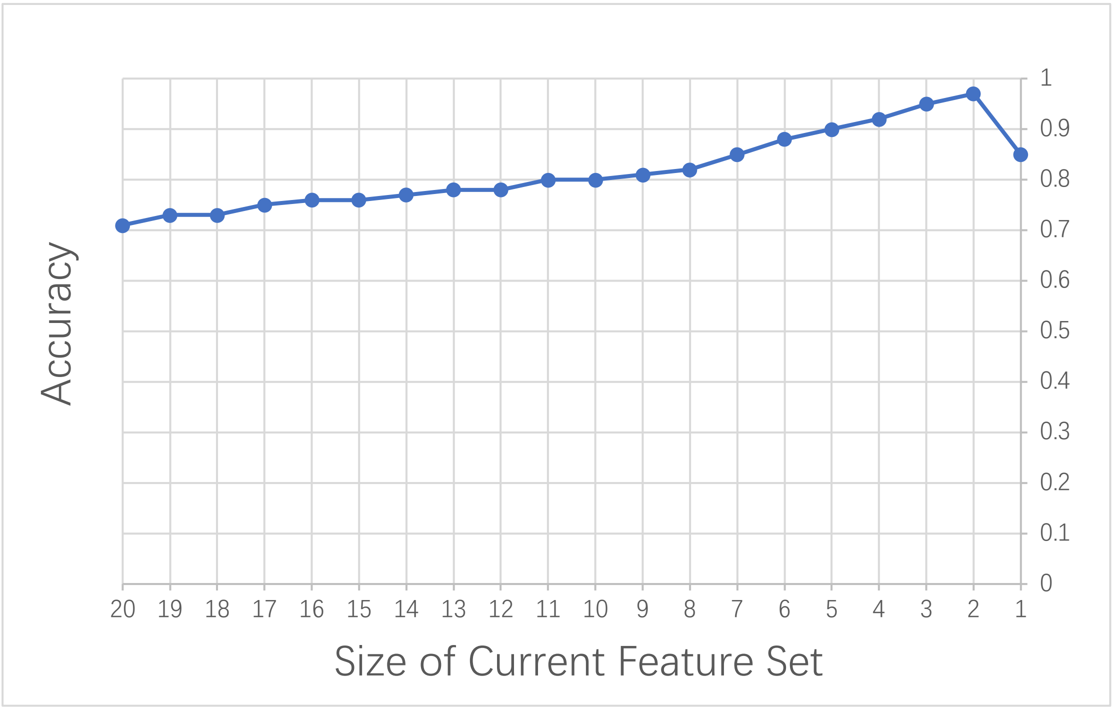

Figure 4 Backward Elimination on CS170_large_Data__1

Comparison of the running time

Instance	Forward Selection	Backward Elimination
CS170_small_Data__1	0.70	0.78
CS170_large_Data__1	15.63	20.02

Table 1 :running time of Forward selection and backward elimination with the large/small datasets.

By observing the table, we can find that the forward Selection can get the result faster than Backward Elimination does.

Trace example

Below is a single trace of my algorithm. I am only showing Forward Selection on the small dataset.

```
➜  Project2-new ./feature_selection
Please enter the name of the file to test:
data_sets/CS170_small_Data__1.txt
Enter '0' to use Forward Selection, '1' to use Backward Elimination:
0
Finish parse file:
    Number of instances: 1000
    Number of features: 10
Start forward selection
    Try add feature 0, accuracy is 0.70
    Try add feature 1, accuracy is 0.75
    Try add feature 2, accuracy is 0.70
    Try add feature 3, accuracy is 0.71
    Try add feature 4, accuracy is 0.69
    Try add feature 5, accuracy is 0.71
    Try add feature 6, accuracy is 0.72
    Try add feature 7, accuracy is 0.85
    Try add feature 8, accuracy is 0.68
    Try add feature 9, accuracy is 0.69
Best feature to add: 7, now feature set is {7}
    Try add feature 0, accuracy is 0.84
    Try add feature 1, accuracy is 0.97
    Try add feature 2, accuracy is 0.83
    Try add feature 3, accuracy is 0.83
    Try add feature 4, accuracy is 0.82
    Try add feature 5, accuracy is 0.81
    Try add feature 6, accuracy is 0.82
    Try add feature 8, accuracy is 0.81
    Try add feature 9, accuracy is 0.84
Best feature to add: 1, now feature set is {1, 7}
    Try add feature 0, accuracy is 0.94
    Try add feature 2, accuracy is 0.93
    Try add feature 3, accuracy is 0.93
```

```
        Try add feature 4, accuracy is 0.92
        Try add feature 5, accuracy is 0.93
        Try add feature 6, accuracy is 0.93
        Try add feature 8, accuracy is 0.93
        Try add feature 9, accuracy is 0.94
Best feature to add: 9, now feature set is {1, 7, 9}
        Try add feature 0, accuracy is 0.91
        Try add feature 2, accuracy is 0.91
        Try add feature 3, accuracy is 0.91
        Try add feature 4, accuracy is 0.89
        Try add feature 5, accuracy is 0.90
        Try add feature 6, accuracy is 0.89
        Try add feature 8, accuracy is 0.90
Best feature to add: 0, now feature set is {0, 1, 7, 9}
        Try add feature 2, accuracy is 0.89
        Try add feature 3, accuracy is 0.89
        Try add feature 4, accuracy is 0.88
        Try add feature 5, accuracy is 0.91
        Try add feature 6, accuracy is 0.86
        Try add feature 8, accuracy is 0.88
Best feature to add: 5, now feature set is {0, 1, 5, 7, 9}
        Try add feature 2, accuracy is 0.85
        Try add feature 3, accuracy is 0.87
        Try add feature 4, accuracy is 0.86
        Try add feature 6, accuracy is 0.84
        Try add feature 8, accuracy is 0.85
Best feature to add: 3, now feature set is {0, 1, 3, 5, 7, 9}
        Try add feature 2, accuracy is 0.84
        Try add feature 4, accuracy is 0.82
        Try add feature 6, accuracy is 0.83
        Try add feature 8, accuracy is 0.81
Best feature to add: 2, now feature set is {0, 1, 2, 3, 5, 7, 9}
        Try add feature 4, accuracy is 0.81
        Try add feature 6, accuracy is 0.83
        Try add feature 8, accuracy is 0.79
Best feature to add: 6, now feature set is {0, 1, 2, 3, 5, 6, 7, 9}
        Try add feature 4, accuracy is 0.81
        Try add feature 8, accuracy is 0.79
Best feature to add: 4, now feature set is {0, 1, 2, 3, 4, 5, 6, 7, 9}
        Try add feature 8, accuracy is 0.78
Best feature to add: 8, now feature set is {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Finish foward selection, the best accuracy is 0.97, best feature set is {1, 7}
Time taken by program: 698 milliseconds
```

My code

feature_selection.hpp

```
struct classifier {
int instances, features; // number of instances, number of features
    vector<double> label; //label of each instance
    vector<vector<double>> feature; //features of each instance

    classifier(): instances(0), features(0) { }

    void parse_file(char *filename);
    void forward_selection();
    void backward_elimination();
    double cross_validation(int *current_features); // Calculate the accuracy of the current
feature set
};
```

feature_selection.cpp

```
double classifier::cross_validation(int *current_features) { // Calculate the accuracy of the
current feature set
    int corrects = 0;
    for (int i = 0; i < instances; i++) {
        double min_dist = 1e18;
        int nearest = -1;
        for (int j = 0; j < instances; j++) { // Find the nearest node
            if (i == j) continue;
            double dist = 0;
            for (int k = 0; k < features; k++)
                if (current_features[k]) dist += pow(feature[i][k] - feature[j][k], 2);
            dist = sqrt(dist);
            if (dist < min_dist) {
                min_dist = dist, nearest = j;
            }
        }
        if (label[i] == label[nearest]) ++corrects; // Compare the nearest pair
    }
    return 1.0 * corrects / instances;
}
```

```
void classifier::forward_selection() {
    printf("Start forward selection\n");
    double best_acc = 0;
    int* current_features = new int[features];
    int* best_features = new int[features];
    for (int i = 0; i < features; i++) // Initialize current_features as an empty set
        current_features[i] = 0;
    for (int i = 0; i < features; i++) {
        double max_acc = 0;
        int select_feature = -1;
        for (int j = 0; j < features; j++) {
            if (current_features[j]) continue;
            current_features[j] = 1; // try add j to current_features and check
            double acc = cross_validation(current_features);
            current_features[j] = 0;
            if (acc > max_acc) {
                max_acc = acc, select_feature = j;
            }
        }
        current_features[select_feature] = 1; // add the optimal feature to current_features
        if (max_acc > best_acc) {
            best_acc = max_acc;
            for (int j = 0; j < features; j++)
                best_features[j] = current_features[j];
        }
    }
}

void classifier::backward_elimination() {
    printf("Start backward_elimination\n");
    double best_acc = 0;
    int* current_features = new int[features];
    int* best_features = new int[features];
    for (int i = 0; i < features; i++) // Initialize current_features as a complete set
        current_features[i] = 1;
    for (int i = 0; i < features; i++) {
        double max_acc = 0;
        int select_feature = -1;
        for (int j = 0; j < features; j++) {
            if (!current_features[j]) continue;
            current_features[j] = 0; // try eliminate j from current_features and check
            double acc = cross_validation(current_features);
            current_features[j] = 1;
            if (acc > max_acc) {
                max_acc = acc, select_feature = j;
            }
        }
        current_features[select_feature] = 0; // eliminate the worst feature from current_features
        if (max_acc > best_acc) {
            best_acc = max_acc;
            for (int j = 0; j < features; j++)
                best_features[j] = current_features[j];
        }
    }
}
```