Project 2: Feature Selection with Nearest Neighbor

Student Name: Sidharth Ramkumar SID: 862129657

Solution:

Dataset	Best Feature Set	Accuracy
Small Number: 58	Forward Selection = {6, 3}	96.97%
	Backward Elimination = {1, 2, 3, 4}	77.78%
	Custom Algorithm = Not implemented	-
Large Number: 58	Forward Selection = {16, 17}	96.1%
	Backward Elimination = {} (empty set)	85.19%
	Custom Algorithm = Not implemented	-

----------<Begin Report>----------

In completing this project, I consulted following resources:

1) https://stackoverflow.com/questions/31495311/clion-c-cant-read-open-txt-file-in-project-directory/31495580#31495580?newreg=36f01ae5f3e042cb93206670f0b62cc5
2) https://stackoverflow.com/questions/46663046/save-read-double-vector-from-file-c
3) https://towardsdatascience.com/how-to-calculate-the-mean-and-standard-deviation-normalizing-datasets-in-pytorch-704bd7d05f4c#:~:text=The%20data%20can%20be%20normalized,This%20results%20in%20faster%20convergence.

I. Introduction

For this project I implemented a basic feature search algorithm, nearest neighbor classifier, and leave-one-out method to test rudimentary machine learning algorithms on datasets. To implement this I used the C++ programming language and referenced several mathematical concepts such as the distance formula, default rate, and normalization of data. Algother, my program is now able to successfully execute forward selection and backward elimination over a variable size dataset to consider the best possible features.

II. Challenges

The most challenging aspect of this project was working with CLion and using C++ to manage 2D vectors. I felt a lot of the complexity in my code resulted from using the C++ language to iterate through each dataset repeatedly for every feature which almost bogged down the runtime of the program. On top of that, CLion ends up crashing when the runtime lags or a lack of response from the compiler is provided, so it makes it difficult to run and test programs. I was able to optimize my program to run through small datasets in quick time, but large datasets take upwards of 10 minutes to complete.

III. Code Design

https://github.com/sidrk01/CS170-Nearest_Neighbor.git

My design utilized 2 different classes along with a main.cpp and uifunctions.h header file.

For my classes I included a SearchAlgos class that contains forward selection and backward elimination as well as the accuracy and nearest neighbor methods to implement the leave-one-out method and train the machine learning algorithm.

My other class was Normalize, which essentially grabs a 2D vector object and normalizes the data using various methods I implemented to calculate relevant statistics like mean, standard deviation, and distance formula.

Optimizations:

One optimization I included for my program is the use of iterators to assist in traversing data quickly as opposed to the regular for-loop. I also utilized the built-in 2D vector copy, and the use of stringstream data type to parse lines of string instantaneously. These allowed me to speed up the execution time of my program and reduce latency with my search algorithms.

IV. Dataset details

The General Small Dataset: Number of features, number of instances

Total Features: 10

Instances: 100

The General Large Dataset: Number of features, number of instances

Total Features: 40

Instances: 1000

Your Small Dataset: Number of features, number of instances

Total Features: 10

Instances: 100

Your Large Dataset: Number of features, number of instances

Total Features: 40

Instances: 1000

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

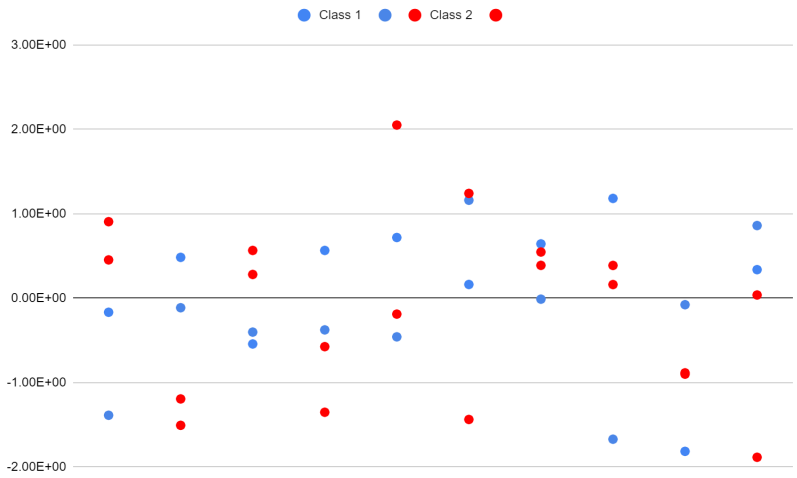
V. Algorithms

1. Forward Selection

The forward selection algorithm is a greedy algorithm responsible for traversing a tree with possible combinations of features and searching for the one with the highest accuracy. The algorithm traverses the search space and takes into account a validation function (leave-one-out-cross-validation). Then it greedily selects the best feature and tests combinations of two features and greedily picks a combination and so on. It stores previously selected features in memory and leaves the rest and goes back when the accuracy has decreased from a selected combination.

2. Backward Elimination

The backward elimination algorithm is also a greedy algorithm that is responsible for traversing a search tree starting with all possible features and eliminating one by one to reach the highest point of accuracy. It passes in all the features into the validation function and then removes combinations with the lowest algorithm. It is very similar to forward selection since it greedily eliminates features that reduce accuracy.



This dataset from my personal small data indicates extraneous features that can be restricted to allow for a more accurate nearest neighbors classification of the dataset.

VI. Analysis

Experiment 1: Comparing Forward Selection vs Backward Elimination.

Compare accuracy with no feature selection vs with feature selection.

The accuracy of my small data is 83% with the default rate (no features). The accuracy with the correct features is {6,3} is 96.97%.

The accuracy of my large data using the default rate is 85.1%. The accuracy with the best features {16,17} is 96.1%.

There is a **sharp increase in the accuracy** when using feature selection.

Compare feature set and accuracy for forward selection vs backward elimination.

The accuracy of my small data is 77.78% using backward elimination and 96.97% for my forward selection.

The accuracy of the general small data is 79.8% using backward elimination and 92.93% accuracy using forward selection

There seems to be a **decline in accuracy** when looking at the backward elimination vs the forward selection.

Talk about pros and cons of both algorithms.

Some pros of using forward selection is the increase in accuracy and the ability to retrace its steps and select other combinations of algorithms. A con is the speed of forward selection and its latency.

A pro with backward selection is the speed and the quickness in selection of features, a major con is the dip in accuracy. Once a feature is eliminated, it is gone from the search.

Experiment 2: Effect of normalization

Compare accuracy when using normalized vs unnormalized data.

The accuracy of my small data is 96.97% using normalized forward selection and 95.96% for my not normalized forward selection.

The accuracy of my large data is 96.91% using normalized forward selection and 96.1% for my not normalized forward selection.

There is a **minor but visible difference in accuracy** for normalizing the data for any given dataset vs. not normalizing the data when utilizing the search algorithms.

Experiment 3: Effect of number neighbors (k)

Plot accuracy vs increasing values of k and examine the trend.

VII. Conclusion

General summary of your findings from the Analysis. Potential improvements to this approach of doing feature selection.

Through the experimentation and analysis of my datasets and machine learning/search algorithms I noticed quite a few factors:

Using features selected by my algorithms will always produce better accuracy. Forward selection is marginally better than my backward selection algorithm but the latency increases as well. Finally, the normalization of data contributes towards increasing the accuracy of the selected features.

VIII. Trace of your small dataset

Utilizing Forward Selection algorithm:

```
Welcome to Sid's Feature Selection Algorithm.

================================================================================
```

```
Please include the filename you would like to test
sid_small.txt
================================================================================================
Would you like to normalize the continuous features?
Enter 'Y' to confirm.
y
[1] Forward Selection
[2] Backward Elimination
1
Using no features, I get an accuracy of 83%

Beginning search.

Using feature(s) {1} accuracy is 73.74%
Using feature(s) {2} accuracy is 76.77%
Using feature(s) {3} accuracy is 70.71%
Using feature(s) {4} accuracy is 73.74%
Using feature(s) {5} accuracy is 69.7%
Using feature(s) {6} accuracy is 87.88%
Using feature(s) {7} accuracy is 74.75%
Using feature(s) {8} accuracy is 76.77%
Using feature(s) {9} accuracy is 67.68%
Using feature(s) {10} accuracy is 76.77%
Feature set {6} was best, accuracy is 87.88%

Using feature(s) {6,1} accuracy is 86.87%
Using feature(s) {6,2} accuracy is 86.87%
Using feature(s) {6,3} accuracy is 96.97%
Using feature(s) {6,4} accuracy is 80.81%
Using feature(s) {6,5} accuracy is 87.88%
Using feature(s) {6,7} accuracy is 82.83%
Using feature(s) {6,8} accuracy is 77.78%
Using feature(s) {6,9} accuracy is 83.84%
Using feature(s) {6,10} accuracy is 70.71%
Feature set {6,3} was best, accuracy is 96.97%

Using feature(s) {6,3,1} accuracy is 90.91%
Using feature(s) {6,3,2} accuracy is 94.95%
Using feature(s) {6,3,4} accuracy is 87.88%
Using feature(s) {6,3,5} accuracy is 84.85%
Using feature(s) {6,3,7} accuracy is 89.9%
Using feature(s) {6,3,8} accuracy is 93.94%
Using feature(s) {6,3,9} accuracy is 91.92%
Using feature(s) {6,3,10} accuracy is 83.84%
Feature set {6,3,2} was best, accuracy is 94.95%

(Warning, Accuracy has decreased!)
Using feature(s) {6,3,2,1} accuracy is 93.94%
Using feature(s) {6,3,2,4} accuracy is 83.84%
Using feature(s) {6,3,2,5} accuracy is 87.88%
Using feature(s) {6,3,2,7} accuracy is 91.92%
Using feature(s) {6,3,2,8} accuracy is 87.88%
```

```
Using feature(s) {6,3,2,9} accuracy is 85.86%
Using feature(s) {6,3,2,10} accuracy is 80.81%
Feature set {6,3,2,1} was best, accuracy is 93.94%

(Warning, Accuracy has decreased!)
Using feature(s) {6,3,2,1,4} accuracy is 79.8%
Using feature(s) {6,3,2,1,5} accuracy is 86.87%
Using feature(s) {6,3,2,1,7} accuracy is 82.83%
Using feature(s) {6,3,2,1,8} accuracy is 82.83%
Using feature(s) {6,3,2,1,9} accuracy is 83.84%
Using feature(s) {6,3,2,1,10} accuracy is 76.77%
Feature set {6,3,2,1,5} was best, accuracy is 86.87%

(Warning, Accuracy has decreased!)
Using feature(s) {6,3,2,1,5,4} accuracy is 77.78%
Using feature(s) {6,3,2,1,5,7} accuracy is 82.83%
Using feature(s) {6,3,2,1,5,8} accuracy is 78.79%
Using feature(s) {6,3,2,1,5,9} accuracy is 83.84%
Using feature(s) {6,3,2,1,5,10} accuracy is 78.79%
Feature set {6,3,2,1,5,9} was best, accuracy is 83.84%

(Warning, Accuracy has decreased!)
Using feature(s) {6,3,2,1,5,9,4} accuracy is 76.77%
Using feature(s) {6,3,2,1,5,9,7} accuracy is 75.76%
Using feature(s) {6,3,2,1,5,9,8} accuracy is 73.74%
Using feature(s) {6,3,2,1,5,9,10} accuracy is 79.8%
Feature set {6,3,2,1,5,9,10} was best, accuracy is 79.8%

(Warning, Accuracy has decreased!)
Using feature(s) {6,3,2,1,5,9,10,4} accuracy is 77.78%
Using feature(s) {6,3,2,1,5,9,10,7} accuracy is 77.78%
Using feature(s) {6,3,2,1,5,9,10,8} accuracy is 78.79%
Feature set {6,3,2,1,5,9,10,8} was best, accuracy is 78.79%

(Warning, Accuracy has decreased!)
Using feature(s) {6,3,2,1,5,9,10,8,4} accuracy is 70.71%
Using feature(s) {6,3,2,1,5,9,10,8,7} accuracy is 83.84%
Feature set {6,3,2,1,5,9,10,8,7} was best, accuracy is 83.84%

(Warning, Accuracy has decreased!)
Using feature(s) {6,3,2,1,5,9,10,8,7,4} accuracy is 74.75%
Feature set {6,3,2,1,5,9,10,8,7,4} was best, accuracy is 74.75%

(Warning, Accuracy has decreased!)
Finished search!! The best feature subset is {6,3} which has an accuracy of 96.97%
================================================================================
Would you like to try another algorithm?
Enter 'Y' to continue or 'N' to quit.
```

Utilizing Backward Elimination:

```
Welcome to Sid's Feature Selection Algorithm.
```

```
============================================================================================
Please include the filename you would like to test
sid_small.txt
============================================================================================
Would you like to normalize the continuous features?
Enter 'Y' to confirm.
y
[1] Forward Selection
[2] Backward Elimination
2
Using feature(s) {2,3,4,5,6,7,8,9,10} accuracy is 67.68%
Using feature(s) {1,3,4,5,6,7,8,9,10} accuracy is 67.68%
Using feature(s) {1,2,4,5,6,7,8,9,10} accuracy is 67.68%
Using feature(s) {1,2,3,5,6,7,8,9,10} accuracy is 67.68%
Using feature(s) {1,2,3,4,6,7,8,9,10} accuracy is 67.68%
Using feature(s) {1,2,3,4,5,7,8,9,10} accuracy is 67.68%
Using feature(s) {1,2,3,4,5,6,8,9,10} accuracy is 67.68%
Using feature(s) {1,2,3,4,5,6,7,9,10} accuracy is 67.68%
Using feature(s) {1,2,3,4,5,6,7,8,10} accuracy is 67.68%
Using feature(s) {1,2,3,4,5,6,7,8,9} accuracy is 67.68%
Feature set {1,2,3,4,5,6,7,8,9} was best, accuracy is 67.68%

Using feature(s) {2,3,4,5,6,7,8,9} accuracy is 72.73%
Using feature(s) {1,3,4,5,6,7,8,9} accuracy is 72.73%
Using feature(s) {1,2,4,5,6,7,8,9} accuracy is 72.73%
Using feature(s) {1,2,3,5,6,7,8,9} accuracy is 72.73%
Using feature(s) {1,2,3,4,6,7,8,9} accuracy is 72.73%
Using feature(s) {1,2,3,4,5,7,8,9} accuracy is 72.73%
Using feature(s) {1,2,3,4,5,6,8,9} accuracy is 72.73%
Using feature(s) {1,2,3,4,5,6,7,9} accuracy is 72.73%
Using feature(s) {1,2,3,4,5,6,7,8} accuracy is 72.73%
Feature set {1,2,3,4,5,6,7,8} was best, accuracy is 72.73%

Using feature(s) {2,3,4,5,6,7,8} accuracy is 75.76%
Using feature(s) {1,3,4,5,6,7,8} accuracy is 75.76%
Using feature(s) {1,2,4,5,6,7,8} accuracy is 75.76%
Using feature(s) {1,2,3,5,6,7,8} accuracy is 75.76%
Using feature(s) {1,2,3,4,6,7,8} accuracy is 75.76%
Using feature(s) {1,2,3,4,5,7,8} accuracy is 75.76%
Using feature(s) {1,2,3,4,5,6,8} accuracy is 75.76%
Using feature(s) {1,2,3,4,5,6,7} accuracy is 75.76%
Feature set {1,2,3,4,5,6,7} was best, accuracy is 75.76%

Using feature(s) {2,3,4,5,6,7} accuracy is 74.75%
Using feature(s) {1,3,4,5,6,7} accuracy is 74.75%
Using feature(s) {1,2,4,5,6,7} accuracy is 74.75%
Using feature(s) {1,2,3,5,6,7} accuracy is 74.75%
Using feature(s) {1,2,3,4,6,7} accuracy is 74.75%
Using feature(s) {1,2,3,4,5,7} accuracy is 74.75%
Using feature(s) {1,2,3,4,5,6} accuracy is 74.75%
Feature set {1,2,3,4,5,6} was best, accuracy is 74.75%
```

```
(Warning, Accuracy has decreased!)
Using feature(s) {2,3,4,5,6} accuracy is 74.75%
Using feature(s) {1,3,4,5,6} accuracy is 74.75%
Using feature(s) {1,2,4,5,6} accuracy is 74.75%
Using feature(s) {1,2,3,5,6} accuracy is 74.75%
Using feature(s) {1,2,3,4,6} accuracy is 74.75%
Using feature(s) {1,2,3,4,5} accuracy is 74.75%
Feature set {1,2,3,4,5} was best, accuracy is 74.75%

(Warning, Accuracy has decreased!)
Using feature(s) {2,3,4,5} accuracy is 77.78%
Using feature(s) {1,3,4,5} accuracy is 77.78%
Using feature(s) {1,2,4,5} accuracy is 77.78%
Using feature(s) {1,2,3,5} accuracy is 77.78%
Using feature(s) {1,2,3,4} accuracy is 77.78%
Feature set {1,2,3,4} was best, accuracy is 77.78%

Using feature(s) {2,3,4} accuracy is 74.75%
Using feature(s) {1,3,4} accuracy is 74.75%
Using feature(s) {1,2,4} accuracy is 74.75%
Using feature(s) {1,2,3} accuracy is 74.75%
Feature set {1,2,3} was best, accuracy is 74.75%

(Warning, Accuracy has decreased!)
Using feature(s) {2,3} accuracy is 71.72%
Using feature(s) {1,3} accuracy is 71.72%
Using feature(s) {1,2} accuracy is 71.72%
Feature set {1,2} was best, accuracy is 71.72%

(Warning, Accuracy has decreased!)
Using feature(s) {2} accuracy is 73.74%
Using feature(s) {1} accuracy is 73.74%
Feature set {1} was best, accuracy is 73.74%

(Warning, Accuracy has decreased!)
Using feature(s) {} accuracy is 16.16%
Feature set {} was best, accuracy is 16.16%

(Warning, Accuracy has decreased!)
Finished search!! The best feature subset is {1,2,3,4} which has an accuracy of 77.78%
================================================================================
Would you like to try another algorithm?
Enter 'Y' to continue or 'N' to quit.
n

Process finished with exit code 0
```