Project 2: Feature Selection with Nearest Neighbor

Student Name: Jordan Sam SID: 862148753

Solution: 151

Dataset	Best Feature Set	Accuracy
Small Number: 151	Forward Selection = {}	0.93
	Backward Elimination = {1 5 6 7 8 10}	0.91
	Custom Algorithm = Not implemented	N/A
Large Number: 151	Forward Selection = {}	0.836
	Backward Elimination = {1 3 4 5 6 7 8 9 10 13 14 15 16 19 20 21 23 24 26 27 28 29 30 31 32 33 35 36 37 38 39 40}	0.763
	Custom Algorithm = Not implemented	N/A

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

In completing this project, I consulted following resources:

https://www.tutorialspoint.com/mean-and-median-of-a-matrix-in-cplusplus

https://www.tutorialspoint.com/cplusplus-program-to-calculate-standard-deviation\

https://www.statology.org/z-score-normalization/

https://stats.stackexchange.com/questions/495357/why-do-we-normalize-test-data-on-the-parameters-of-the-training-data

https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/#:~:text=Special%20Values%3A%20IEEE%20has%20reserved,though%20they%20both%20are%20equal.

https://www.cplusplus.com/reference/string/stod/

https://www.cplusplus.com/reference/string/stoi/

I also talked about normalizing data, validator class, and classifier class with my partner Kevin Gao.

I. Introduction

This project report is about implementing and using Greedy search, nearest neighbor classifier, and Evaluation using leave-one-out validation on a given dataset. I chose to do this project in C++ because I am more familiar with this language. My progress was kept track of through my github commits at https://github.com/Jsam88/Feature-Selection-Machine-Learning. Running on mac, the command should be "g++ -std=c++11 main.cpp" followed by "./a.out".

II. Challenges

One of the challenges that I ran into was converting the string of data into a double so that the program could run calculations. In order to get the program to properly work I had to do more research on changing the IEEE string. Another challenge for me was normalizing the data. I was not entirely sure on how to find the means or standard deviation of the matrices since I used a vector of vectors to hold data, but after researching more I was also able to solve this problem. My final major challenge was finding bugs. Although I was testing as I was going, I ran into a problem at the end when I was incorporating part 1 and 2 together. Even though I was able to make both part 1 and part 2 of the project work separately, I was really struggling with part 3. After going through a bunch of variables and making sure they were being set to the correct value, I noticed that my number of instances were not being passed into the validator correctly. After some time, I saw that I called my validator at the top of my main before I normalized the data. I was essentially not passing anything into the validator class which caused many bugs in determining the accuracy itself.

III. Code Design

I utilized object oriented programming with a multitude of classes and functions.

DataNormalizer - Function that normalizes the data by taking in the file, opens the file, and passes by references a vector of vectors that hold the training instances and a vector of the training labels.

StringToNum - Function that takes a string passed in from the data file that is formatted as IEEE and changes the value into a double.

GreedySearch - Implements forward selection and backwards elimination for feature selection.

Classifier - Uses the nearest neighbor classifier, train, and test function for data. In order to predict the label, we use euclidean distance which is implemented in the node class.

Node - We store the children in an array and we have the helper functions that push back a child or push back a feature. The function evaluation is done in this class and I updated the randomizer to utilize the validator.

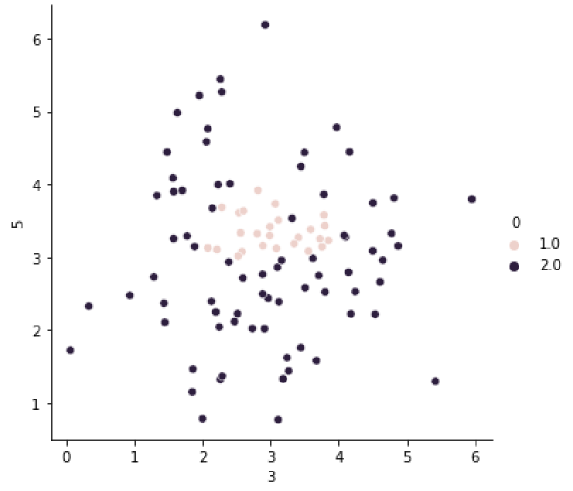
Validator - K-fold testing is done here by using the classifier class. It also has the leave-one-out validation which gives the accuracy to nodes.

Main - Calls on the functions to put everything together (Datanormalizer, classifiers, validator, etc.)

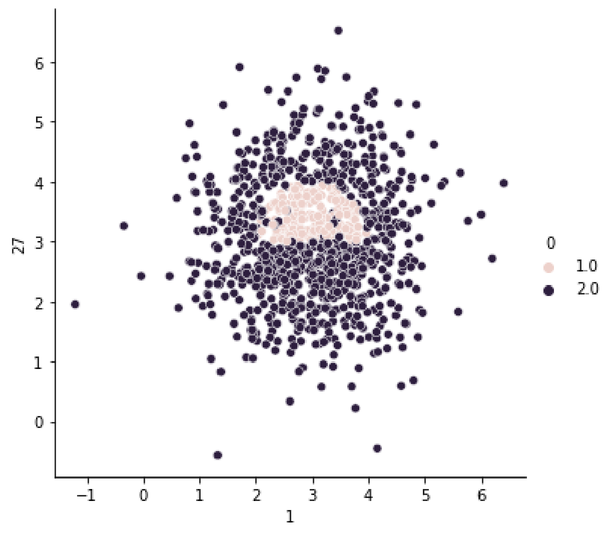
IV. Dataset details

Small Personal Dataset: 10 features, 100 instances
Large Personal Dataset: 40 features, 1000 instances
Since my datasets do not include features, I decided to plot the small and large sample datasets given in part 2 of the project instructions.

Small Sample Dataset: 10 features, 100 instances
Feature 3 and 5 give the best accuracy of around 89%. Here on the graph we graph feature 3 along the x axis and feature 5 along the y axis. Class 1 is color coded pink whereas class 2 is color coded purple.



Large Sample Dataset: 40 features, 1000 instances
Features 1 and 27 give the best accuracy of around 95%. Here on the graph we graph feature 1 along the x axis and feature 27 along the y axis. Class 1 is color coded pink whereas class 2 is color coded purple.



V. Algorithms

1. Forward Selection - iterative algorithm where we initially start with 0 features in the model. We greedily add the feature that has the best improvement/accuracy for our model and once our accuracy starts to decrease, we stop adding features.
2. Backward Elimination - basically forward selection except working backwards. In this case we start off with all of the features implemented and then remove one feature at a time. The subset of features that does the best is kept and we keep removing from there.
3. Jordan’s custom algorithm (optional) - N/A

VI. Analysis

Experiment 1: Comparing Forward Selection vs Backward Elimination.

Forward selection performed better than backward elimination in both sample and personal datasets. Starting with backwards elimination, we are testing all of the different possible combinations which takes more time and power. Running backwards elimination took around 20 minutes to run whereas forwards selection took a minute for my program. This is expected because the nearest neighbor classifier algorithm is still sensitive to irrelevant features. Starting with all the features does not help the time and memory storage in this case. Finally, the accuracy of forward selection is better than backward elimination because we are essentially only adding features that we need while also disregarding the irrelevant ones.

Experiment 2: Effect of normalization.

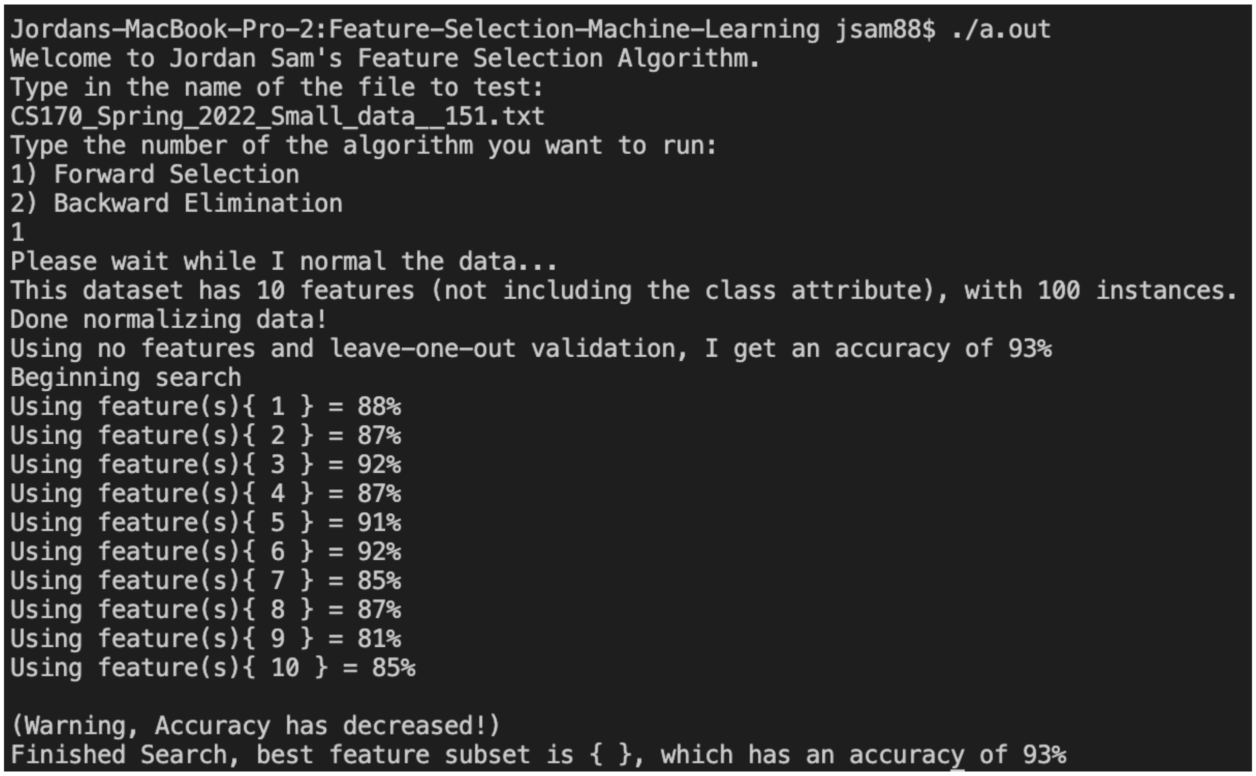
After running my program with my data normalizer functions commented out the forward selection stayed the same and backwards elimination improved. The features in my data set changed and the final print statement was, “Finished Search, best feature subset is { 3 4 7 9 10 }, which has an accuracy of 93%”. The accuracy went up by 2 percent. Overall, not much has changed to the accuracy and this may be due to the data set already being fairly normalized.

VII. Conclusion

In my Feature-Selection-Machine-Learning program, forwards selection performed better compared to backwards elimination. Since we are dealing with accuracy, using only relevant features is very important. By sacrificing the best answer using the greedy search algorithm, we are able to gain efficiency in return. There were small ways I thought about optimizing my code such as removing some calculations to save time and space but that's very minimal overhead. A potential way to improve this approach of feature selection is to use more than 1 k-neighbor.

VIII. Trace of your small dataset

Forward:



```
Jordans-MacBook-Pro-2:Feature-Selection-Machine-Learning jsam88$ ./a.out
Welcome to Jordan Sam's Feature Selection Algorithm.
Type in the name of the file to test:
CS170_Spring_2022_Small_data__151.txt
Type the number of the algorithm you want to run:
1) Forward Selection
2) Backward Elimination
1
Please wait while I normal the data...
This dataset has 10 features (not including the class attribute), with 100 instances.
Done normalizing data!
Using no features and leave-one-out validation, I get an accuracy of 93%
Beginning search
Using feature(s){ 1 } = 88%
Using feature(s){ 2 } = 87%
Using feature(s){ 3 } = 92%
Using feature(s){ 4 } = 87%
Using feature(s){ 5 } = 91%
Using feature(s){ 6 } = 92%
Using feature(s){ 7 } = 85%
Using feature(s){ 8 } = 87%
Using feature(s){ 9 } = 81%
Using feature(s){ 10 } = 85%

(Warning, Accuracy has decreased!)
Finished Search, best feature subset is { }, which has an accuracy of 93%
```

Backward:

Welcome to Jordan Sam's Feature Selection Algorithm.

Type in the name of the file to test:

CS170_Spring_2022_Small_data__151.txt

Type the number of the algorithm you want to run:

1) Forward Selection

2) Backward Elimination

2

Please wait while I normal the data...

This dataset has 10 features (not including the class attribute), with 100 instances.

Done normalizing data!

Using all features and leave-one-out validation, I get an accuracy of 87%

Beginning search

```
Using feature(s){ 2 3 4 5 6 7 8 9 10 } = 85%
Using feature(s){ 1 3 4 5 6 7 8 9 10 } = 88%
Using feature(s){ 1 2 4 5 6 7 8 9 10 } = 87%
Using feature(s){ 1 2 3 5 6 7 8 9 10 } = 86%
Using feature(s){ 1 2 3 4 6 7 8 9 10 } = 83%
Using feature(s){ 1 2 3 4 5 7 8 9 10 } = 85%
Using feature(s){ 1 2 3 4 5 6 8 9 10 } = 82%
Using feature(s){ 1 2 3 4 5 6 7 9 10 } = 86%
Using feature(s){ 1 2 3 4 5 6 7 8 10 } = 88%
Using feature(s){ 1 2 3 4 5 6 7 8 9 } = 87%

Feature set { 1 3 4 5 6 7 8 9 10 } was best, accuracy is 88%

Using feature(s){ 3 4 5 6 7 8 9 10 } = 85%
Using feature(s){ 1 4 5 6 7 8 9 10 } = 89%
Using feature(s){ 1 3 5 6 7 8 9 10 } = 84%
Using feature(s){ 1 3 4 6 7 8 9 10 } = 85%
Using feature(s){ 1 3 4 5 7 8 9 10 } = 88%
Using feature(s){ 1 3 4 5 6 8 9 10 } = 85%
Using feature(s){ 1 3 4 5 6 7 9 10 } = 89%
Using feature(s){ 1 3 4 5 6 7 8 10 } = 87%
Using feature(s){ 1 3 4 5 6 7 8 9 } = 87%

Feature set { 1 4 5 6 7 8 9 10 } was best, accuracy is 89%

Using feature(s){ 4 5 6 7 8 9 10 } = 88%
Using feature(s){ 1 5 6 7 8 9 10 } = 87%
Using feature(s){ 1 4 6 7 8 9 10 } = 84%
Using feature(s){ 1 4 5 7 8 9 10 } = 88%
Using feature(s){ 1 4 5 6 8 9 10 } = 88%
Using feature(s){ 1 4 5 6 7 9 10 } = 88%
Using feature(s){ 1 4 5 6 7 8 10 } = 90%
Using feature(s){ 1 4 5 6 7 8 9 } = 86%

Feature set { 1 4 5 6 7 8 10 } was best, accuracy is 90%


Using feature(s){ 4 5 6 7 8 10 } = 88%
Using feature(s){ 1 5 6 7 8 10 } = 91%
Using feature(s){ 1 4 6 7 8 10 } = 89%
Using feature(s){ 1 4 5 7 8 10 } = 90%
Using feature(s){ 1 4 5 6 8 10 } = 86%
Using feature(s){ 1 4 5 6 7 10 } = 89%
Using feature(s){ 1 4 5 6 7 8 } = 88%

Feature set { 1 5 6 7 8 10 } was best, accuracy is 91%


Using feature(s){ 5 6 7 8 10 } = 88%
Using feature(s){ 1 6 7 8 10 } = 88%
Using feature(s){ 1 5 7 8 10 } = 90%
Using feature(s){ 1 5 6 8 10 } = 87%
Using feature(s){ 1 5 6 7 10 } = 87%
Using feature(s){ 1 5 6 7 8 } = 90%

(Warning, Accuracy has decreased!)
Finished Search, best feature subset is { 1 5 6 7 8 10 }, which has an accuracy of 91%
```