Phillip Nguyen
862050241

CS170 Project #2 Report

Objective/Introduction:

This project's purpose is to implement a feature selection algorithm using nearest neighbor. We are given two types of datasets, a small dataset and a large dataset. We use these datasets to find the best features for classification. To find these feature subsets, we implemented two greedy algorithms, which are forward selection and backward elimination. These algorithms return a subset of features with the highest accuracy found. Since these algorithms are greedy, local maximums will affect the results. This program is implemented with Python 3.7 and the traces are provided at the end of the report.

Design:

- Used numpy array for certain parts of normalization to try to speed up program
- 2d list to store the data from the text file
- Converted this list to a numpy 2d array, to use some utility functions when normalizing
    - The utility functions used were finding the mean and the standard deviation
        - The only time numpy was used for the final code
        - The actually normalization was done by me
    - Attempted to use numpy array to speed up math calculations
        - Math calculations are faster through numpy, attempted to speed up in smaller areas to improve times
    - Converted numpy array back to list for quicker access
- Used as much list comprehension when making lists to marginally speed up program
- Used lists to keep track of features and subset of features
- Used copy library to perform deep copy on lists
- Nearest neighbor is based on single nearest neighbor

- Validator implemented is “leave one out”
- Greedy forward selection search
    - Starts with empty set and works until you find set with highest accuracy
    - Stops when accuracy starts dropping, it is a greedy algorithm
- Greedy backward selection search
    - Starts with full set of features and removes one feature at a time
    - Finds set with highest accuracy and stops when accuracy decreases
- The code has option for exhaustive search just in case these searches only find a local max

Functionality:

- Analyze datasets with greedy forward & backward selection algorithms
- Outputs a trace during the analysis
    - What feature subset its analyzing currently and the accuracy
    - Shows when accuracy is decreasing
    - Shows best choice of subset when moving to next depth of features
- Allows user to enter name of dataset, find the file in directory
- ****NOTE**:** Not sure what the trace looked like with greedy search and not exhaustive
    - Still printed out the warning that accuracy decreased and the subset with the best decreased accuracy, and then it ends

Data & Analysis:

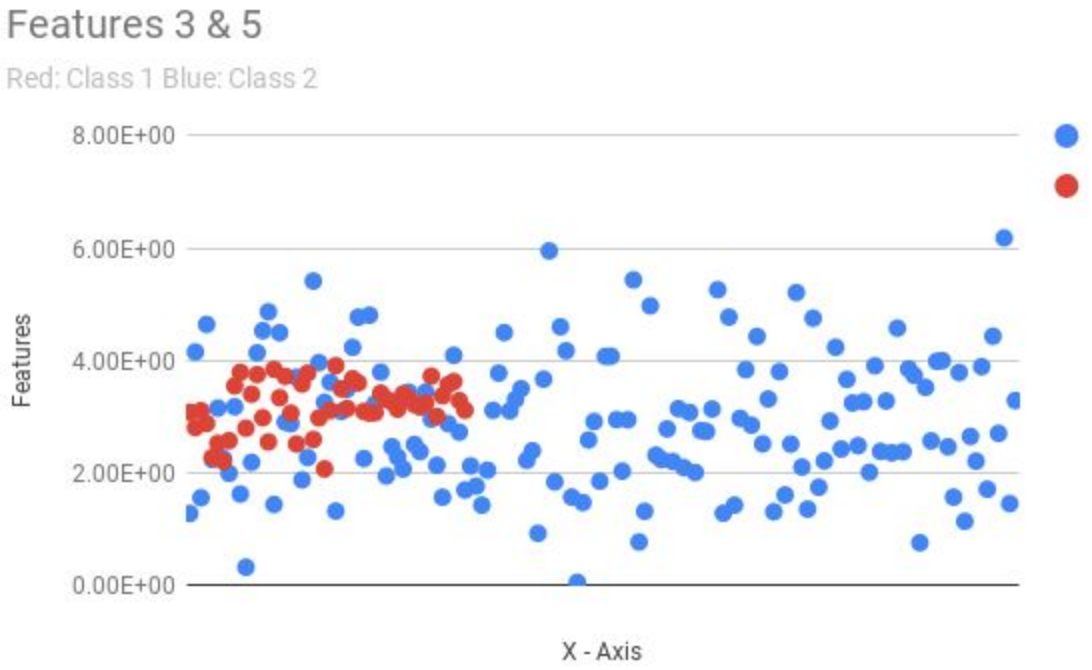
Dataset	Algorithm	Feature Subset	Best Accuracy
Small80	Forward	{5, 3}	92%
	Backward	{2, 4, 5, 7, 10}	82%
Large80	Forward	{27, 1}	95.5%
	Backward	{2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40}	72.2%
Small76	Forward	{10,6}	95%
	Backward	{1, 2, 4, 6, 9, 10}	83%
Large76	Forward	{19, 22}	96.7%
	Backward	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40}	71.6%
Small76	Backwards exhaustive	{2,10}	92%

Analysis of Data:

Due to implementing greedy forward/backward search. Which means the search terminates when the accuracy drops, it is very easily affected by local maxima. As we can see above, forward and backward methods differ in their results when looking for the best subset of features in the dataset. We see that through each test, the forward search always results in a smaller subset than backwards with higher accuracy. This is because forward adds one at a time, and backwards removes them. With backwards removing them, the accuracy dips and increases in many situations, and since this is a greedy algorithm, the backwards search gets stuck in those local maximums very easily.

I ran an exhaustive search on my personal dataset backwards, and got a different answer to what the greedy algorithm provides. Note that I did not run the exhaustive backwards search on all of them, since my run time was very long, I could not sustain running it for so long. I ran the single test for an example of a situation that the backwards search gets stuck in local maxima, as you can see that when going backwards, the greedy choice eventually removes the six, which causes the subset of the features at the end to be not optimal.

Alongside with backwards search not giving optimal results, from my code execution, it also takes longer. Because it starts with such a massive set of features to test, and slowly removes one by one, it takes a long time. Again since it is greedy, it will choose the highest accuracy, so it gets stuck early on in the stage of removing features and gives back a suboptimal solution set.



Looking at the graph above, graphing the feature subset {3,5} from the small80 dataset on a scatter plot. With Blue being class 2 points and red being class 1 points. We can see that class 2 seems to surround class 1 in terms of the shape of the data points. That means it is more likely that the nearest neighbor algorithm will classify correctly, since there is separation between the classes. Again there are errors since the data points are not separated cleanly.

Conclusion:

From the datasets that we worked with on this project. Using the single nearest neighbor algorithm alongside the leave one out validator, and implementing greedy forward and backward search algorithms. The forward search method returns the highest accuracy feature subset more than the backwards search. The accuracy of the forward search was always higher than backwards and the feature subset was smaller. To be clear, these findings do not conclude that either search method is better since it is

tested on very few datasets. But from this project, and the data received on the provided datasets, forward & backward search don't always return the same value due to their greedy nature, and they do not always return the most optimal solution since they are greedy. From the data, forward search provided more accurate feature subsets than backwards search.

Furthermore, through observation, backwards search did take longer than forward search due to having a bigger starting subset of features to look through. Additionally, my self-implemented nearest neighbor could have been faster, compared to the libraries provided out there on the Internet.

Challenges:

- Biggest challenge was speed of the program
    - Tried to use numpy library to speed up certain parts of code
        - Calculations are done faster through numpy than normal python
        - Bottleneck is found in nearest neighbor and unsorted and non indexed dataset
    - Attempted to use list comprehension and faster insertion and deletion methods where I thought would improve parts of code
    - Removed square root calculation in euclidean distance calculation
        - Did not need actual value, just used it for comparison so the results did not change
    - Convert numpy array back to list because numpy assessors were slower than native python list
    - Due to greedy nature of algorithm, I broke the search early, program would have been running for a lot longer if the search was exhaustive
- Understanding the way the algorithm classified data
    - I had a hard time at the beginning figuring out the search algorithm
        - The nearest neighbor and leave one out validator was simpler

- I used the slides that professor provided to help with search algorithms

Improvements in the Future:

- Incorporate numpy for nearest neighbor
  - Attempted to use numpy's euclidean distance calculation method because its much faster
  - Would have to find a different way to access the data in other places
- Sort and index data
  - Having to search through data made it very slow
  - Try to hash as much data as possible for faster access
- Improve trace program
  - Due to nature of trace in guidelines, I felt it was easier to print while doing the search
  - The console outputs can slow down program quite a bit
  - In the future, possible sacrifice space and increase complexity of code for more speed

Sources:

- Normalization: https://www.codecademy.com/articles/normalization
- Nearest Neighbor: https://www.dataquest.io/blog/k-nearest-neighbors-in-python/
  - Used this link mostly for euclidean distance in nearest neighbor
  - Slides & Lecture helped more with nearest neighbor pseudocode
- Numpy: https://numpy.org/doc/
- Python: https://docs.python.org/3/

- ****NOTE**: NUMPY was installed through pycharm**
  - **Python 3.7**
  - **Pycharm 2020.1**
  - **Numpy: 1.18**

- **Installing numpy on pycharm: https://stackoverflow.com/questions/35623776/import-numpy-on-pycharm**

Trace: (Personal dataset renamed to small2.txt)

```
C:\Users\iiNza\Documents\Python\python.exe "C:/Users/iiNza/Documents/Code/CS170/Project 2/cs170.py"
Welcome to Phillip Nguyen's Feature Selection Algorithm.
Type in the name of the file to test: small2.txt
This dataset has 10 features(not including the class attribute), with 100 instances.
Please wait while I normalize the data... Done!
Type the number of the algorithm you want to run.
1) Forward Selection
2) Backward Elimination
1
Running nearest neighbor with all 100 features, using "leave one out evaluation", I get an accuracy of 67.0%
Beginning search.
Using feature(s) [1] accuracy is 64.0%
Using feature(s) [2] accuracy is 64.0%
Using feature(s) [3] accuracy is 66.0%
Using feature(s) [4] accuracy is 57.99999999999999%
Using feature(s) [5] accuracy is 69.0%
Using feature(s) [6] accuracy is 71.0%
Using feature(s) [7] accuracy is 64.0%
Using feature(s) [8] accuracy is 76.0%
Using feature(s) [9] accuracy is 67.0%
Using feature(s) [10] accuracy is 86.0%
Feature set [10] was best, accuracy is 86.0%
Using feature(s) [10, 1] accuracy is 84.0%
Using feature(s) [10, 2] accuracy is 92.0%
Using feature(s) [10, 3] accuracy is 85.0%
Using feature(s) [10, 4] accuracy is 82.0%
Using feature(s) [10, 5] accuracy is 85.0%
Using feature(s) [10, 6] accuracy is 95.0%
Using feature(s) [10, 7] accuracy is 81.0%
Using feature(s) [10, 8] accuracy is 84.0%
Using feature(s) [10, 9] accuracy is 82.0%
Feature set [10, 6] was best, accuracy is 95.0%
Using feature(s) [10, 6, 1] accuracy is 85.0%
Using feature(s) [10, 6, 2] accuracy is 88.0%
Using feature(s) [10, 6, 3] accuracy is 88.0%
Using feature(s) [10, 6, 4] accuracy is 85.0%
Using feature(s) [10, 6, 5] accuracy is 91.0%
Using feature(s) [10, 6, 7] accuracy is 91.0%
Using feature(s) [10, 6, 8] accuracy is 79.0%
Using feature(s) [10, 6, 9] accuracy is 84.0%
(Warning, Accuracy has decreased! Stopping Search)
Feature set [10, 6, 5] was best, accuracy is 91.0%
Finished search!! The best feature subset is [10, 6] which has an accuracy of 95.0%

Process finished with exit code 0
```

```
C:\Users\iiNza\Documents\Python\python.exe "C:/Users/iiNza/Documents/Code/CS170/Project 2/cs170.py"
Welcome to Phillip Nguyen's Feature Selection Algorithm.
Type in the name of the file to test: small2.txt
This dataset has 10 features(not including the class attribute), with 100 instances.
Please wait while I normalize the data... Done!
Type the number of the algorithm you want to run.
1) Forward Selection
2) Backward Elimination
2
Running nearest neighbor with all 100 features, using "leave one out evaluation", I get an accuracy of 67.0%
Beginning search.
Using feature(s) [2, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is 67.0%
Using feature(s) [1, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is 63.0%
Using feature(s) [1, 2, 4, 5, 6, 7, 8, 9, 10] accuracy is 74.0%
Using feature(s) [1, 2, 3, 5, 6, 7, 8, 9, 10] accuracy is 69.0%
Using feature(s) [1, 2, 3, 4, 6, 7, 8, 9, 10] accuracy is 74.0%
Using feature(s) [1, 2, 3, 4, 5, 7, 8, 9, 10] accuracy is 71.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 8, 9, 10] accuracy is 71.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 7, 9, 10] accuracy is 76.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 7, 8, 10] accuracy is 75.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 7, 8, 9] accuracy is 68.0%
Feature set [1, 2, 3, 4, 5, 6, 7, 9, 10] was best, accuracy is 76.0%
Using feature(s) [2, 3, 4, 5, 6, 7, 9, 10] accuracy is 70.0%
Using feature(s) [1, 3, 4, 5, 6, 7, 9, 10] accuracy is 73.0%
Using feature(s) [1, 2, 4, 5, 6, 7, 9, 10] accuracy is 71.0%
Using feature(s) [1, 2, 3, 5, 6, 7, 9, 10] accuracy is 76.0%
Using feature(s) [1, 2, 3, 4, 6, 7, 9, 10] accuracy is 77.0%
Using feature(s) [1, 2, 3, 4, 5, 7, 9, 10] accuracy is 72.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 9, 10] accuracy is 71.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 7, 10] accuracy is 76.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 7, 9] accuracy is 67.0%
Feature set [1, 2, 3, 4, 6, 7, 9, 10] was best, accuracy is 77.0%
Using feature(s) [2, 3, 4, 6, 7, 9, 10] accuracy is 76.0%
Using feature(s) [1, 3, 4, 6, 7, 9, 10] accuracy is 70.0%
Using feature(s) [1, 2, 4, 6, 7, 9, 10] accuracy is 81.0%
Using feature(s) [1, 2, 3, 6, 7, 9, 10] accuracy is 70.0%
Using feature(s) [1, 2, 3, 4, 7, 9, 10] accuracy is 77.0%
Using feature(s) [1, 2, 3, 4, 6, 9, 10] accuracy is 76.0%
Using feature(s) [1, 2, 3, 4, 6, 7, 10] accuracy is 81.0%
Using feature(s) [1, 2, 3, 4, 6, 7, 9] accuracy is 73.0%
Feature set [1, 2, 4, 6, 7, 9, 10] was best, accuracy is 81.0%
Using feature(s) [2, 4, 6, 7, 9, 10] accuracy is 72.0%
Using feature(s) [1, 4, 6, 7, 9, 10] accuracy is 73.0%
Using feature(s) [1, 2, 6, 7, 9, 10] accuracy is 69.0%
Using feature(s) [1, 2, 4, 7, 9, 10] accuracy is 80.0%
Using feature(s) [1, 2, 4, 6, 9, 10] accuracy is 83.0%
Using feature(s) [1, 2, 4, 6, 7, 10] accuracy is 79.0%
Using feature(s) [1, 2, 4, 6, 7, 9] accuracy is 78.0%
Feature set [1, 2, 4, 6, 9, 10] was best, accuracy is 83.0%
Using feature(s) [2, 4, 6, 9, 10] accuracy is 75.0%
Using feature(s) [1, 4, 6, 9, 10] accuracy is 75.0%
Using feature(s) [1, 2, 6, 9, 10] accuracy is 75.0%
Using feature(s) [1, 2, 4, 9, 10] accuracy is 79.0%
Using feature(s) [1, 2, 4, 6, 10] accuracy is 81.0%
Using feature(s) [1, 2, 4, 6, 9] accuracy is 72.0%
(Warning, Accuracy has decreased! Stopping Search)
Feature set [1, 2, 4, 6, 10] was best, accuracy is 81.0%
Finished search!! The best feature subset is [1, 2, 4, 6, 9, 10] which has an accuracy of 83.0%
```