Project 2: Feature Selection with Nearest Neighbor

Name: Shaoyu Tu
SID: 862137500
Email: stu024@ucr.edu
Date: May 03, 2024
Github Link (https://github.com/Kevin20201/CS_205)

The following report's format takes inspiration from Dr. Eamonn Keogh's sample report. [1].

Below is a list of links and short description attached that I consulted in order to complete this project.

- NumPy Documentation:
  - Referenced NumPy unique to return the predicted class: https://numpy.org/doc/stable/reference/generated/numpy.unique.html
  - Referenced NumPy argsort to sort by index of distances: https://numpy.org/doc/stable/reference/generated/numpy.argsort.html
- The Python Library to understand how to drop column: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html
- Professor Eamon Keogh's slide deck 6__MachineLearning001 slide 66 and 47 for pseudocode of KNN algorithm and K-fold cross validation respectively
- Professor Eamon Keogh's Greedy Forward feature selection algorithm walkthrough from Slide deck 7__MachineLearning002 slide 25-31

All major parts in KNN, greedy forward feature search, and greedy backward feature search code is original. Subroutines borrowed from Python libraries includes...

- Subroutines from `numpy`: `unique()` function is used to return the nearest neighbors' classes for prediction.
- Subroutines from `numpy`: `argsort()` function is used to sort by index given the distance list.
- Subroutines from `numpy`: `sqrt()` function is used to take the squareroot of a number for the Euclidean distance algorithm.
- Subroutines from `numpy`: `concat()` function is used to append the greedily selected feature in the forward method to the existing list.
- Subroutines from `copy`: `deepcopy()` function is used to distinguish between copy by value from the standard copy by reference.

"*I affirm that I did not use ChatGPT or similar to write the code or text in this work.*" [1]

Contents

Objective

This project aims to run greedy forward and greedy backward feature selection based on leave-one out cross validation. Specifically, our objective is to implement the K Nearest Neighbor algorithm and both feature selection process to then evaluate the feature set that gives us the best class separation.

Evaluation

Below are two graphs that shows us the features selected from each of the datasets assigned to me. The first section discusses my result from the small dataset assigned to me CS205_small_Data__21.txt. The second section discusses my result from the large dataset assigned to me CS205_large_Data__13.txt.

CS205_small_Data__21.txt

Forward Search

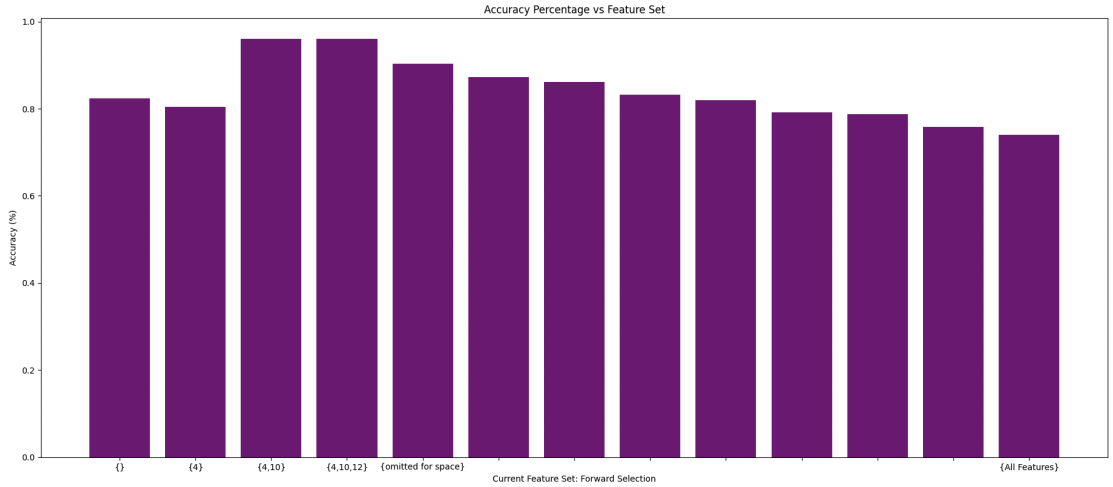
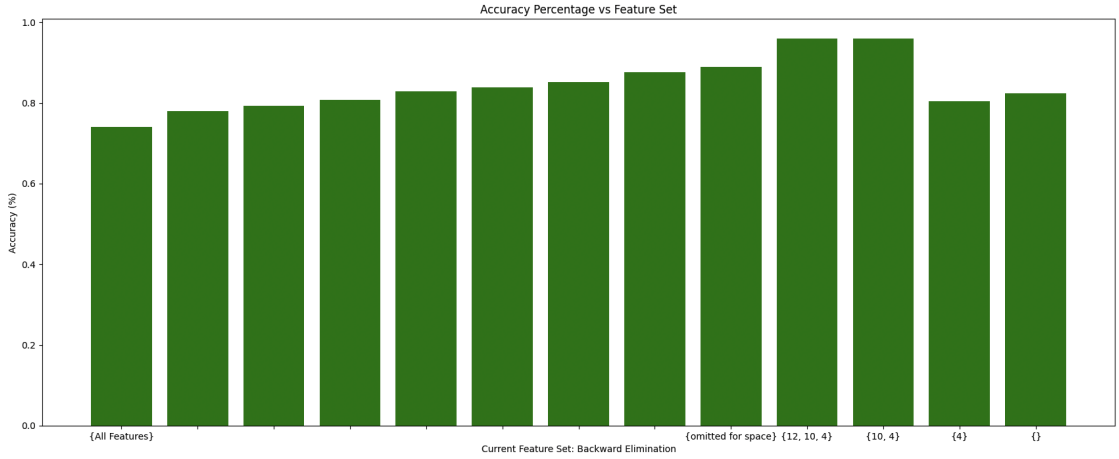


Figure 1: *Accuracy of increasingly large subsets of features discovered by forward selection on the small dataset.*

In Figure 1 we begin by running based on an empty feature set and reported it as the default rate. My reported default rate is `82.4%`. Then we iterate through all available features and test them one by one and return the feature that results in the highest accuracy. In my case adding feature `4` gives us the highest accuracy at `80.4%`. Surprisingly we see a drop in accuracy with adding in our first feature, but this can be explained as a result of my imbalance in classes. Following feature 4, we then test the next best feature by adding all other features to the existing feature set and as a result we find adding feature `10` gives us the highest accuracy. The accuracy from feature set `{4, 10}` gives us `96%`. Then adding another feature we get feature set `{4, 10, 12}` also returning `96%` accuracy which does not improve accuracy. Then, continuing on we add feature `8` to the existing feature sets and this time we actually see a decrease in accuracy down to `90.4%`. Finally, this process will continue until all features are in the list, and we will see accuracy continues to decrease down to `74%`.

The best feature subset with forward selection was features `{4,10}` with accuracy `96%`.

Backward Search



) Figure 2: *Accuracy of increasingly large subsets of features discovered by backward selection.*

In Figure 2, we look at the feature selection but now with backwards elimination. Reversing this feature selection process, we first do a sanity check for the accuracy when all features are used which we get `74%`. This result is what we have previously gotten from the forward search which is good. Then, we iterate over with removing one feature from the feature set and notice when feature `7` is removed, our accuracy will go up to `78%`. We continue to do this elimination and returning highest accuracy until 3 features are left in the set, which we see is `{12, 10, 4}` from the graph. This is the same as forward search and returns an accuracy of `96%`. Then, with 2 features left over we again got features `{10, 4}` by removing feature 12 and accuracy of `96%`. Finally, our single best feature was again feature `4` which tells us that feature `4` can be trusted to be a good feature to work with. The accuracy of only using feature 4 was again `80.4%`. However, this result is strange as it returns to us the unique case where our forward selection and backwards elimination resulted in the same features to select.

The best feature subset with backwards elimination was features `{12, 10, 4}` with accuracy `96%`.

Conclusion for Small Dataset

From the results of feature selction and backward elimination feature search, I belive that the best features for this small dataset assigned to me are features `4, 10 and 12`. Due to both feature `10` and `12` returning an accuracy of `96%`, either in this case could interchange their order of being selected. Thus, there weren't really a weak feature. If these features, features `{4, 10, 12}`, were chosen for modelling, then I would think the accuracy will be around `96%`.

CS205_large_Data__13.txt

Forward Search

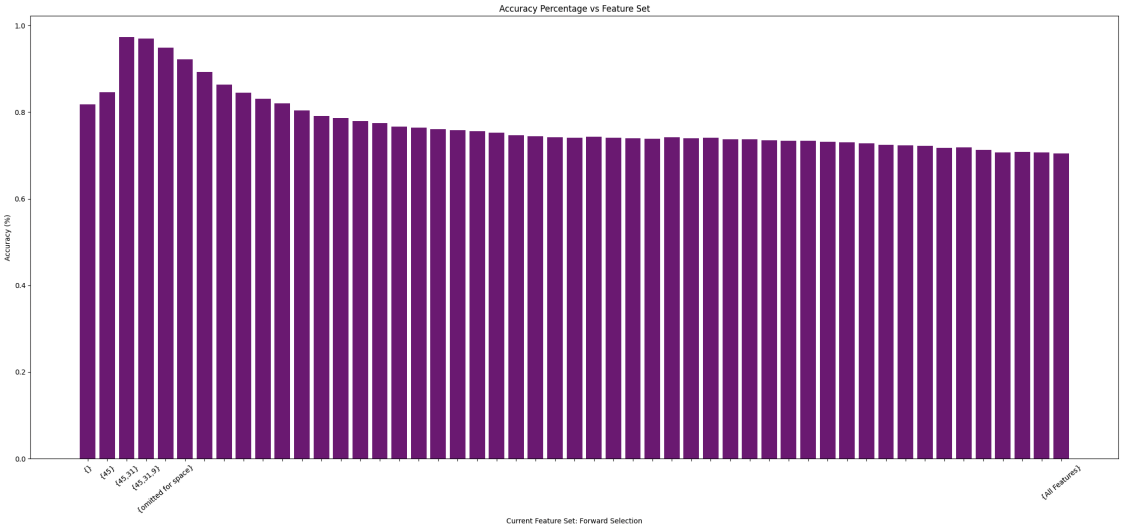


Figure 3: *Accuracy of increasingly large subsets of features discovered by forward selection on the large dataset.*

We will now analyze our result outputted from running the forward feature selection on the assigned large dataset (Large Dataset 13). From looking at Figure 3, we also begin with computing our default rate from an empty feature set. The default rate we obtained is `81.84%`. Adding our first best feature greedily, we add feature `45` who returns us an accuracy of `84.58%`. This time however, compared to the small dataset, we do see an increase in the accuracy but still notices the large imbalance of the classes in our dataset. We continue to search for a second best feature and end up with feature set `{45, 31}` where feature `31` is added and bumps the accuracy up dramatically to `97.36%`. The third feature we add is feature `9` and this is where our accuracy begins to drop with an accuracy of `97.02%`. Finally, if you take a glance at the plot, you will notice this trend of accuracy decreasing until all features are in the feature set leaving us with an accuracy of `70.44%`.

The best feature subset with forward selection was features `{45,31}` with accuracy `97.36%`.

Backward Search

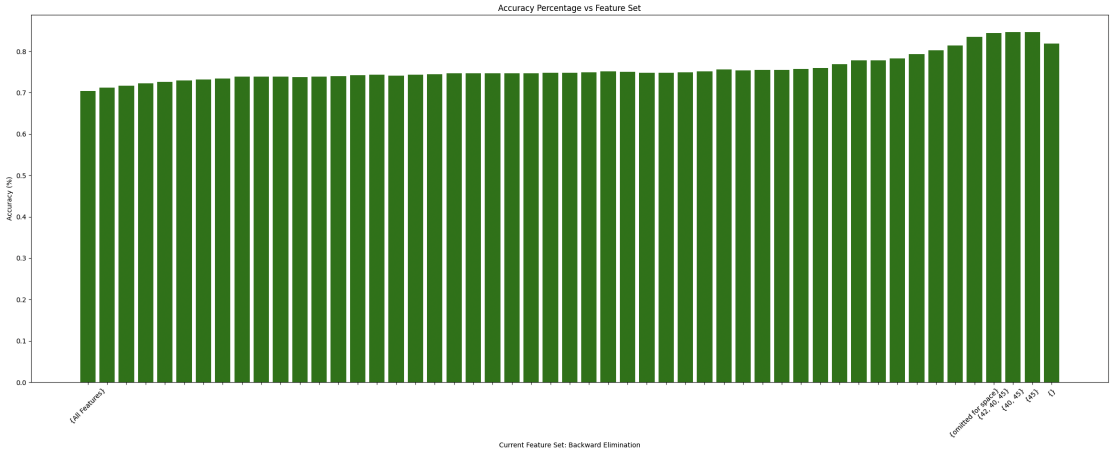


Figure 4: *Accuracy of increasingly large subsets of features discovered by backward elimination on the large dataset.*

Similar to the backwards elimination process for small dataset, we start with a sanity check with all features used which gives us an accuracy of `70.44%`. This matches the case in forward selection so we are ready to continue. Again, we look to eliminate features and we note that the first feature we chose to eliminate was feature `11`. By removing this feature we increase our accuracy to `71.18%`. Then continuing on until we have 3 features left, the feature set we have is `{42, 40, 45}`. This feature set gives has an accuracy of `84.4%`. Then, the next iteration we remove feature `42` and are left with features `{40, 45}` where we get an accuracy of `84.58%`. Finally, the last feature we are left with is feature `45` with an accuracy of again `84.58%`. This tells us that for this large dataset feature `45` is a really good feature as it doubles up with the result from forward selection.

The best feature subset with backwards elimination was features `{40, 45}` with accuracy `84.58%`.

Conclusion for Large Dataset

From the results of feature selction and backward elimination feature search, I belive that the best features for this large dataset assigned to me is feature `45`. As for a second and third choice, I would go with features `31 and 9` as the weak features for this dataset that was chosen by feature selection. This is because forward selection generally better than backward elimination, and the accuracies presented by forward selection for having 2 features and 3 features in the feature subset also resulted in high accuracy as `97.36%` and `97.02%` respectively. Compared to backward elimination, when feature subset had 2 and 3 features remain in the feature list, our model was only performing at around `84%`. Thus, I would pick these features, features `{45, 31, 9}`, for modelling, and I would think the accuracy will be around `97%`.

Computational Effort for Search

For my implementation, the code is presented in Python as this is my language of choice for most things machine learning. I ran all my experiments on my M2 Macbook Pro laptop with a Apple M2 Pro chip and 16GB of main memory. The table below shows my runtime for the 4 searches that was completed for this project.

	Small 21 Dataset (12 features, 500 instances)	Large 13 Dataset (50 features, 5000 instances)
Forward Selection	31.9 sec	1 day 27 min 8 sec (88,028 sec recorded)
Backward Elimination	36.8 sec	1 day 12 hours 10 min 45 sec (130,245 sec recorded)

Note that these times were measured at the start of the feature search and ends right after the prints. These times may vary depending on the system used.

Trace

The Small 21 Dataset traces are shown in full. The Large 13 Dataset traces are only partly shown but can be found on my Github following [Github Link](https://github.com/Kevin20201/CS_205).

Forward Selection Small 21 Dataset

The following is a traceback from running Forward Selection on the Small 21 Dataset.

```
Welcome to KNN Feature Selection Algorithm!

Please select the file you would like to test:
1. Sample Small 19
2. Assigned Small 21
3. Sample Large 6
4. Assigned Large 13

Please select the algorithm you would like to run:

1. Forward Selection
2. Backward Elimination
3. Both

This dataset has 12 features (not including the class attribute), with 500 instances.

Running KNN with no features, I get an accuracy of 82.39999999999999%

Beginning search.

    Using feature(s) {1} accuracy is 73.0%
    Using feature(s) {2} accuracy is 71.2%
    Using feature(s) {3} accuracy is 71.0%
    Using feature(s) {4} accuracy is 80.4%
    Using feature(s) {5} accuracy is 71.6%
    Using feature(s) {6} accuracy is 68.2%
    Using feature(s) {7} accuracy is 67.80000000000001%
    Using feature(s) {8} accuracy is 67.80000000000001%
    Using feature(s) {9} accuracy is 69.6%
    Using feature(s) {10} accuracy is 75.0%
    Using feature(s) {11} accuracy is 70.0%
```

```
    Using feature(s) {12} accuracy is 73.6%

Feature set {4} was best, accuracy is 80.4%

    Using feature(s) {4,1} accuracy is 84.0%
    Using feature(s) {4,2} accuracy is 79.2%
    Using feature(s) {4,3} accuracy is 80.0%
    Using feature(s) {4,5} accuracy is 81.39999999999999%
    Using feature(s) {4,6} accuracy is 81.0%
    Using feature(s) {4,7} accuracy is 83.2%
    Using feature(s) {4,8} accuracy is 84.0%
    Using feature(s) {4,9} accuracy is 81.39999999999999%
    Using feature(s) {4,10} accuracy is 96.0%
    Using feature(s) {4,11} accuracy is 82.39999999999999%
    Using feature(s) {4,12} accuracy is 83.8%

Feature set {4,10} was best, accuracy is 96.0%

    Using feature(s) {4,10,1} accuracy is 93.0%
    Using feature(s) {4,10,2} accuracy is 91.60000000000001%
    Using feature(s) {4,10,3} accuracy is 91.0%
    Using feature(s) {4,10,5} accuracy is 93.2%
    Using feature(s) {4,10,6} accuracy is 92.0%
    Using feature(s) {4,10,7} accuracy is 90.0%
    Using feature(s) {4,10,8} accuracy is 91.4%
    Using feature(s) {4,10,9} accuracy is 92.60000000000001%
    Using feature(s) {4,10,11} accuracy is 92.60000000000001%
    Using feature(s) {4,10,12} accuracy is 96.0%

Feature set {4,10,12} was best, accuracy is 96.0%

    Using feature(s) {4,10,12,1} accuracy is 89.8%
    Using feature(s) {4,10,12,2} accuracy is 89.60000000000001%
    Using feature(s) {4,10,12,3} accuracy is 87.6%
    Using feature(s) {4,10,12,5} accuracy is 88.2%
    Using feature(s) {4,10,12,6} accuracy is 89.0%
    Using feature(s) {4,10,12,7} accuracy is 89.0%
    Using feature(s) {4,10,12,8} accuracy is 90.4%
    Using feature(s) {4,10,12,9} accuracy is 88.6%
    Using feature(s) {4,10,12,11} accuracy is 86.6%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8} was best, accuracy is 90.4%

    Using feature(s) {4,10,12,8,1} accuracy is 86.4%
    Using feature(s) {4,10,12,8,2} accuracy is 87.2%
    Using feature(s) {4,10,12,8,3} accuracy is 87.0%
    Using feature(s) {4,10,12,8,5} accuracy is 86.0%
    Using feature(s) {4,10,12,8,6} accuracy is 85.39999999999999%
    Using feature(s) {4,10,12,8,7} accuracy is 86.2%
    Using feature(s) {4,10,12,8,9} accuracy is 84.6%
```

```
    Using feature(s) {4,10,12,8,11} accuracy is 86.8%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8,2} was best, accuracy is 87.2%

    Using feature(s) {4,10,12,8,2,1} accuracy is 83.6%
    Using feature(s) {4,10,12,8,2,3} accuracy is 84.0%
    Using feature(s) {4,10,12,8,2,5} accuracy is 83.2%
    Using feature(s) {4,10,12,8,2,6} accuracy is 86.2%
    Using feature(s) {4,10,12,8,2,7} accuracy is 83.0%
    Using feature(s) {4,10,12,8,2,9} accuracy is 82.39999999999999%
    Using feature(s) {4,10,12,8,2,11} accuracy is 85.2%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8,2,6} was best, accuracy is 86.2%

    Using feature(s) {4,10,12,8,2,6,1} accuracy is 83.2%
    Using feature(s) {4,10,12,8,2,6,3} accuracy is 80.80000000000001%
    Using feature(s) {4,10,12,8,2,6,5} accuracy is 81.0%
    Using feature(s) {4,10,12,8,2,6,7} accuracy is 80.80000000000001%
    Using feature(s) {4,10,12,8,2,6,9} accuracy is 81.0%
    Using feature(s) {4,10,12,8,2,6,11} accuracy is 83.2%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8,2,6,11} was best, accuracy is 83.2%

    Using feature(s) {4,10,12,8,2,6,11,1} accuracy is 81.2%
    Using feature(s) {4,10,12,8,2,6,11,3} accuracy is 82.0%
    Using feature(s) {4,10,12,8,2,6,11,5} accuracy is 80.0%
    Using feature(s) {4,10,12,8,2,6,11,7} accuracy is 79.80000000000001%
    Using feature(s) {4,10,12,8,2,6,11,9} accuracy is 81.0%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8,2,6,11,3} was best, accuracy is 82.0%

    Using feature(s) {4,10,12,8,2,6,11,3,1} accuracy is 79.2%
    Using feature(s) {4,10,12,8,2,6,11,3,5} accuracy is 79.2%
    Using feature(s) {4,10,12,8,2,6,11,3,7} accuracy is 78.4%
    Using feature(s) {4,10,12,8,2,6,11,3,9} accuracy is 79.2%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8,2,6,11,3,9} was best, accuracy is 79.2%

    Using feature(s) {4,10,12,8,2,6,11,3,9,1} accuracy is 78.60000000000001%
    Using feature(s) {4,10,12,8,2,6,11,3,9,5} accuracy is 77.4%
    Using feature(s) {4,10,12,8,2,6,11,3,9,7} accuracy is 78.8%
```

```
(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8,2,6,11,3,9,7} was best, accuracy is 78.8%

    Using feature(s) {4,10,12,8,2,6,11,3,9,7,1} accuracy is 75.8%
    Using feature(s) {4,10,12,8,2,6,11,3,9,7,5} accuracy is 73.6%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8,2,6,11,3,9,7,1} was best, accuracy is 75.8%

    Using feature(s) {4,10,12,8,2,6,11,3,9,7,1,5} accuracy is 74.0%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {4,10,12,8,2,6,11,3,9,7,1,5} was best, accuracy is 74.0%

Finished search!!! The best feature subset is {4,10}, which has an accuracy of 96.0%
Elapsed (with compilation) = 31.925284147262573
```

Backward Elimination Small 21 Dataset

The following is a traceback from running Backward Elimination on the Small 21 Dataset.

> *Note that the implementation lists the feature subset as features removed instead of features being used. Thus, for the evaluation section I manually took the difference of the subset from a complete feature subset.*

```
Welcome to KNN Feature Selection Algorithm!

Please select the file you would like to test:
1. Sample Small 19
2. Assigned Small 21
3. Sample Large 6
4. Assigned Large 13

Please select the algorithm you would like to run:

1. Forward Selection
2. Backward Elimination
3. Both

This dataset has 12 features (not including the class attribute), with 500 instances.

Running KNN with all 12 features, using "leave-one-out" evaluation, I get an accuracy
of 74.0%

Beginning search.

    Removing feature(s) {1} accuracy is 73.6%
    Removing feature(s) {2} accuracy is 76.0%
    Removing feature(s) {3} accuracy is 75.4%
```

```
    Removing feature(s) {4} accuracy is 70.0%
    Removing feature(s) {5} accuracy is 75.8%
    Removing feature(s) {6} accuracy is 75.2%
    Removing feature(s) {7} accuracy is 78.0%
    Removing feature(s) {8} accuracy is 75.0%
    Removing feature(s) {9} accuracy is 74.0%
    Removing feature(s) {10} accuracy is 71.39999999999999%
    Removing feature(s) {11} accuracy is 72.6%
    Removing feature(s) {12} accuracy is 73.8%

Feature set {7} was best, accuracy is 78.0%

    Removing feature(s) {7,1} accuracy is 77.4%
    Removing feature(s) {7,2} accuracy is 79.2%
    Removing feature(s) {7,3} accuracy is 77.8%
    Removing feature(s) {7,4} accuracy is 72.39999999999999%
    Removing feature(s) {7,5} accuracy is 78.60000000000001%
    Removing feature(s) {7,6} accuracy is 76.8%
    Removing feature(s) {7,8} accuracy is 78.2%
    Removing feature(s) {7,9} accuracy is 77.4%
    Removing feature(s) {7,10} accuracy is 75.2%
    Removing feature(s) {7,11} accuracy is 79.0%
    Removing feature(s) {7,12} accuracy is 76.0%

Feature set {7,2} was best, accuracy is 79.2%

    Removing feature(s) {7,2,1} accuracy is 79.80000000000001%
    Removing feature(s) {7,2,3} accuracy is 79.2%
    Removing feature(s) {7,2,4} accuracy is 73.2%
    Removing feature(s) {7,2,5} accuracy is 80.80000000000001%
    Removing feature(s) {7,2,6} accuracy is 79.2%
    Removing feature(s) {7,2,8} accuracy is 77.4%
    Removing feature(s) {7,2,9} accuracy is 79.0%
    Removing feature(s) {7,2,10} accuracy is 76.2%
    Removing feature(s) {7,2,11} accuracy is 79.0%
    Removing feature(s) {7,2,12} accuracy is 74.4%

Feature set {7,2,5} was best, accuracy is 80.80000000000001%

    Removing feature(s) {7,2,5,1} accuracy is 81.2%
    Removing feature(s) {7,2,5,3} accuracy is 80.0%
    Removing feature(s) {7,2,5,4} accuracy is 74.2%
    Removing feature(s) {7,2,5,6} accuracy is 81.0%
    Removing feature(s) {7,2,5,8} accuracy is 79.80000000000001%
    Removing feature(s) {7,2,5,9} accuracy is 82.8%
    Removing feature(s) {7,2,5,10} accuracy is 78.2%
    Removing feature(s) {7,2,5,11} accuracy is 79.4%
    Removing feature(s) {7,2,5,12} accuracy is 77.2%

Feature set {7,2,5,9} was best, accuracy is 82.8%

    Removing feature(s) {7,2,5,9,1} accuracy is 83.8%
```

```
    Removing feature(s) {7,2,5,9,3} accuracy is 82.8%
    Removing feature(s) {7,2,5,9,4} accuracy is 73.4%
    Removing feature(s) {7,2,5,9,6} accuracy is 82.8%
    Removing feature(s) {7,2,5,9,8} accuracy is 83.39999999999999%
    Removing feature(s) {7,2,5,9,10} accuracy is 77.8%
    Removing feature(s) {7,2,5,9,11} accuracy is 81.39999999999999%
    Removing feature(s) {7,2,5,9,12} accuracy is 82.19999999999999%

Feature set {7,2,5,9,1} was best, accuracy is 83.8%

    Removing feature(s) {7,2,5,9,1,3} accuracy is 85.2%
    Removing feature(s) {7,2,5,9,1,4} accuracy is 75.4%
    Removing feature(s) {7,2,5,9,1,6} accuracy is 84.2%
    Removing feature(s) {7,2,5,9,1,8} accuracy is 84.2%
    Removing feature(s) {7,2,5,9,1,10} accuracy is 79.4%
    Removing feature(s) {7,2,5,9,1,11} accuracy is 83.39999999999999%
    Removing feature(s) {7,2,5,9,1,12} accuracy is 82.39999999999999%

Feature set {7,2,5,9,1,3} was best, accuracy is 85.2%

    Removing feature(s) {7,2,5,9,1,3,4} accuracy is 72.6%
    Removing feature(s) {7,2,5,9,1,3,6} accuracy is 86.8%
    Removing feature(s) {7,2,5,9,1,3,8} accuracy is 87.6%
    Removing feature(s) {7,2,5,9,1,3,10} accuracy is 82.19999999999999%
    Removing feature(s) {7,2,5,9,1,3,11} accuracy is 85.39999999999999%
    Removing feature(s) {7,2,5,9,1,3,12} accuracy is 82.8%

Feature set {7,2,5,9,1,3,8} was best, accuracy is 87.6%

    Removing feature(s) {7,2,5,9,1,3,8,4} accuracy is 71.8%
    Removing feature(s) {7,2,5,9,1,3,8,6} accuracy is 86.6%
    Removing feature(s) {7,2,5,9,1,3,8,10} accuracy is 82.39999999999999%
    Removing feature(s) {7,2,5,9,1,3,8,11} accuracy is 89.0%
    Removing feature(s) {7,2,5,9,1,3,8,12} accuracy is 88.6%

Feature set {7,2,5,9,1,3,8,11} was best, accuracy is 89.0%

    Removing feature(s) {7,2,5,9,1,3,8,11,4} accuracy is 71.39999999999999%
    Removing feature(s) {7,2,5,9,1,3,8,11,6} accuracy is 96.0%
    Removing feature(s) {7,2,5,9,1,3,8,11,10} accuracy is 81.8%
    Removing feature(s) {7,2,5,9,1,3,8,11,12} accuracy is 92.0%

Feature set {7,2,5,9,1,3,8,11,6} was best, accuracy is 96.0%

    Removing feature(s) {7,2,5,9,1,3,8,11,6,4} accuracy is 75.4%
    Removing feature(s) {7,2,5,9,1,3,8,11,6,10} accuracy is 83.8%
    Removing feature(s) {7,2,5,9,1,3,8,11,6,12} accuracy is 96.0%

Feature set {7,2,5,9,1,3,8,11,6,12} was best, accuracy is 96.0%

    Removing feature(s) {7,2,5,9,1,3,8,11,6,12,4} accuracy is 75.0%
    Removing feature(s) {7,2,5,9,1,3,8,11,6,12,10} accuracy is 80.4%
```

```
(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {7,2,5,9,1,3,8,11,6,12,10} was best, accuracy is 80.4%

    Removing feature(s) {7,2,5,9,1,3,8,11,6,12,10,4} accuracy is 82.39999999999999%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set {7,2,5,9,1,3,8,11,6,12,10,4} was best, accuracy is 82.39999999999999%

Finished search!!! The best feature subset is {7,2,5,9,1,3,8,11,6}, which has an
accuracy of 96.0%
Elapsed (after compilation) = 36.85779881477356
```

Forward Selection Large 13 Dataset

The following is a traceback from running Forward Selection on the Large 13 Dataset.

> *Note this is an old trace collected when the accuracy was not yet multiplied by 100. But due to the long runtime, I was unable to make the small changes and rerun for nearly the same output.*

```
Welcome to KNN Feature Selection Algorithm!

Please select the file you would like to test:
1. Sample Small 19
2. Assigned Small 21
3. Sample Large 6
4. Assigned Large 13

Your selection: Please select the algorithm you would like to run:

1. Forward Selection
2. Backward Elimination
3. Both

Your selection: This dataset has 50 features (not including the class attribute), with
5000 instances.

Running KNN with no features, I get an accuracy of 0.8184%

Beginning search.

    Using feature(s) {1} accuracy is 0.7028%
    Using feature(s) {2} accuracy is 0.7016%
    Using feature(s) {3} accuracy is 0.6952%
    Using feature(s) {4} accuracy is 0.704%
    Using feature(s) {5} accuracy is 0.7012%
    Using feature(s) {6} accuracy is 0.7006%
    Using feature(s) {7} accuracy is 0.702%
    Using feature(s) {8} accuracy is 0.6946%
    Using feature(s) {9} accuracy is 0.7206%
```

```
    Using feature(s) {10} accuracy is 0.694%
    Using feature(s) {11} accuracy is 0.702%
    Using feature(s) {12} accuracy is 0.6934%
    Using feature(s) {13} accuracy is 0.7014%
    Using feature(s) {14} accuracy is 0.7058%
    Using feature(s) {15} accuracy is 0.699%
    Using feature(s) {16} accuracy is 0.7076%
    Using feature(s) {17} accuracy is 0.6988%
    Using feature(s) {18} accuracy is 0.713%
    Using feature(s) {19} accuracy is 0.6984%
    Using feature(s) {20} accuracy is 0.6914%
    Using feature(s) {21} accuracy is 0.6964%
    Using feature(s) {22} accuracy is 0.699%
    Using feature(s) {23} accuracy is 0.7072%
    Using feature(s) {24} accuracy is 0.7032%
    Using feature(s) {25} accuracy is 0.7174%
    Using feature(s) {26} accuracy is 0.7096%
    Using feature(s) {27} accuracy is 0.6942%
    Using feature(s) {28} accuracy is 0.7006%
    Using feature(s) {29} accuracy is 0.7028%
    Using feature(s) {30} accuracy is 0.7186%
    Using feature(s) {31} accuracy is 0.7382%
    Using feature(s) {32} accuracy is 0.6994%
    Using feature(s) {33} accuracy is 0.7048%
    Using feature(s) {34} accuracy is 0.6928%
    Using feature(s) {35} accuracy is 0.7042%
    Using feature(s) {36} accuracy is 0.7072%
    Using feature(s) {37} accuracy is 0.7022%
    Using feature(s) {38} accuracy is 0.6976%
    Using feature(s) {39} accuracy is 0.7076%
    Using feature(s) {40} accuracy is 0.705%
    Using feature(s) {41} accuracy is 0.7024%
    Using feature(s) {42} accuracy is 0.7018%
    Using feature(s) {43} accuracy is 0.701%
    Using feature(s) {44} accuracy is 0.7014%
    Using feature(s) {45} accuracy is 0.8458%
    Using feature(s) {46} accuracy is 0.6956%
    Using feature(s) {47} accuracy is 0.7012%
    Using feature(s) {48} accuracy is 0.6966%
    Using feature(s) {49} accuracy is 0.7074%
    Using feature(s) {50} accuracy is 0.7048%

Feature set {45} was best, accuracy is 0.8458%
```

{Professor, I have removed a large chunk of the intermediate result and instead pasted the first and last few levels of the search}

```
Feature set
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 was best, accuracy is 0.7122%

    Using feature(s)
```

```
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7074%
    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7054%
    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7052%
    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7072%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 was best, accuracy is 0.7074%

    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7014%
    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7084%
    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7016%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 was best, accuracy is 0.7084%

    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7074%
    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7016%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 was best, accuracy is 0.7074%

    Using feature(s)
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 accuracy is 0.7044%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)
```

```
Feature set
{45,31,9,21,29,17,43,20,14,33,38,1,47,35,34,3,15,11,23,12,5,25,19,39,8,26,40,28,16,42,2,
 was best, accuracy is 0.7044%

Finished search!!! The best feature subset is {45,31}, which has an accuracy of
0.9736%
Elapsed (with compilation) = 88028.3649020195
```

Backward Elimination Large 13 Dataset

The following is a traceback from running Backward Elimination on the Large 13 Dataset.

> *Note this is an old trace collected when the accuracy was not yet multiplied by 100. But due to the long runtime, I was unable to make the small changes and rerun for nearly the same output. Note that the implementation lists the feature subset as features removed instead of features being used. Thus, for the evaluation section I manually took the difference of the subset from a complete feature subset.*

```
Welcome to KNN Feature Selection Algorithm!

Please select the file you would like to test:
1. Sample Small 19
2. Assigned Small 21
3. Sample Large 6
4. Assigned Large 13

Your selection: Please select the algorithm you would like to run:

1. Forward Selection
2. Backward Elimination
3. Both

Your selection: This dataset has 50 features (not including the class attribute), with
5000 instances.

Running KNN with all 50 features, using "leave-one-out" evaluation, I get an accuracy
of 0.7044%

Beginning search.

    Removing feature(s) {1} accuracy is 0.7052%
    Removing feature(s) {2} accuracy is 0.7024%
    Removing feature(s) {3} accuracy is 0.7034%
    Removing feature(s) {4} accuracy is 0.7082%
    Removing feature(s) {5} accuracy is 0.7046%
    Removing feature(s) {6} accuracy is 0.7082%
    Removing feature(s) {7} accuracy is 0.7024%
    Removing feature(s) {8} accuracy is 0.711%
    Removing feature(s) {9} accuracy is 0.7096%
    Removing feature(s) {10} accuracy is 0.7036%
```

```
    Removing feature(s) {11} accuracy is 0.7118%
    Removing feature(s) {12} accuracy is 0.702%
    Removing feature(s) {13} accuracy is 0.704%
    Removing feature(s) {14} accuracy is 0.7064%
    Removing feature(s) {15} accuracy is 0.7012%
    Removing feature(s) {16} accuracy is 0.701%
    Removing feature(s) {17} accuracy is 0.7042%
    Removing feature(s) {18} accuracy is 0.7094%
    Removing feature(s) {19} accuracy is 0.7052%
    Removing feature(s) {20} accuracy is 0.7004%
    Removing feature(s) {21} accuracy is 0.7026%
    Removing feature(s) {22} accuracy is 0.707%
    Removing feature(s) {23} accuracy is 0.71%
    Removing feature(s) {24} accuracy is 0.6984%
    Removing feature(s) {25} accuracy is 0.7014%
    Removing feature(s) {26} accuracy is 0.7012%
    Removing feature(s) {27} accuracy is 0.7038%
    Removing feature(s) {28} accuracy is 0.6996%
    Removing feature(s) {29} accuracy is 0.7024%
    Removing feature(s) {30} accuracy is 0.7016%
    Removing feature(s) {31} accuracy is 0.7056%
    Removing feature(s) {32} accuracy is 0.6962%
    Removing feature(s) {33} accuracy is 0.706%
    Removing feature(s) {34} accuracy is 0.7004%
    Removing feature(s) {35} accuracy is 0.709%
    Removing feature(s) {36} accuracy is 0.701%
    Removing feature(s) {37} accuracy is 0.707%
    Removing feature(s) {38} accuracy is 0.696%
    Removing feature(s) {39} accuracy is 0.7042%
    Removing feature(s) {40} accuracy is 0.7044%
    Removing feature(s) {41} accuracy is 0.7016%
    Removing feature(s) {42} accuracy is 0.7078%
    Removing feature(s) {43} accuracy is 0.7004%
    Removing feature(s) {44} accuracy is 0.7%
    Removing feature(s) {45} accuracy is 0.7062%
    Removing feature(s) {46} accuracy is 0.7074%
    Removing feature(s) {47} accuracy is 0.7038%
    Removing feature(s) {48} accuracy is 0.6998%
    Removing feature(s) {49} accuracy is 0.7018%
    Removing feature(s) {50} accuracy is 0.7102%

Feature set {11} was best, accuracy is 0.7118%
```

{Professor, I have removed a large chunk of the intermediate result and instead pasted the first and last few levels of the search}

```
Feature set
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 was best, accuracy is 0.8138%

    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
```

```
 accuracy is 0.8248%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.8344%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.8208%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.8194%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.7046%

Feature set
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 was best, accuracy is 0.8344%

    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.844%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.8392%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.8346%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.704%

Feature set
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 was best, accuracy is 0.844%

    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.834%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.8458%
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.7088%

Feature set
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 was best, accuracy is 0.8458%

    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.8458%
```

```
    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.705%

Feature set
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 was best, accuracy is 0.8458%

    Removing feature(s)
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 accuracy is 0.8184%

(Warning, Accuracy has decreased!!! Continuing search in case of local maxima)

Feature set
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 was best, accuracy is 0.8184%

Finished search!!! The best feature subset is
{11,31,50,8,18,33,44,27,39,6,29,32,35,43,5,9,41,7,4,12,38,13,19,22,48,34,2,10,36,30,1,15
 which has an accuracy of 0.8458%
Elapsed (after compilation) = 130245.3639717102
```

Implementation

[Github Link](https://github.com/Kevin20201/CS_205) (https://github.com/Kevin20201/CS_205)

> *Note, the implementation may get cut off on the far right side but can be found on the Github page.*

KNN.py

```
import numpy as np
import pandas as pd
import copy
import time

### Sample Files
### Referenced GeeksforGeeks to read in a txt file https://www.geeksforgeeks.org/how-to-read-space-delimited-files-in-pandas/#
small_data_19 =
pd.read_csv('/Users/kevintu/Documents/Python/CS205/CS_205/CS205_small_Data__19.txt',
sep='  ', header=None)
### Referenced pandas library documentation for renaming a column
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html
small_data_19 = small_data_19.rename(columns={0 : "label"})

### Referenced GeeksforGeeks to read in a txt file https://www.geeksforgeeks.org/how-to-read-space-delimited-files-in-pandas/#
large_data_6 =
pd.read_csv('/Users/kevintu/Documents/Python/CS205/CS_205/CS205_large_Data__6.txt',
sep='  ', header=None)
```

```
### Referenced pandas library documentation for renaming a column
https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.DataFrame.rename.html
large_data_6 = large_data_6.rename(columns={0 : "label"})

### Assigned Files
### Referenced GeeksforGeeks to read in a txt file https://www.geeksforgeeks.org/how-
to-read-space-delimited-files-in-pandas/#
small_data_21 =
pd.read_csv('/Users/kevintu/Documents/Python/CS205/CS_205/CS205_small_Data__21.txt',
sep='  ', header=None)
### Referenced pandas library documentation for renaming a column
https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.DataFrame.rename.html
small_data_21 = small_data_21.rename(columns={0 : "label"})

### Referenced GeeksforGeeks to read in a txt file https://www.geeksforgeeks.org/how-
to-read-space-delimited-files-in-pandas/#
large_data_13 =
pd.read_csv('/Users/kevintu/Documents/Python/CS205/CS_205/CS205_large_Data__13.txt',
sep='  ', header=None)
### Referenced pandas library documentation for renaming a column
https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.DataFrame.rename.html
large_data_13 = large_data_13.rename(columns={0 : "label"})

class KNN_Classifier:
    def __init__(self, train_set=None, test_set=None, k=None, nearest_neighbors=None):
        self.train_set = train_set
        self.test_set = test_set
        self.k = k
        self.nearest_neightbors = nearest_neighbors

    def distance(self, nums):
        sum = 0
        for num in nums:
            sum += num**2
        return np.sqrt(sum)

    def pred_class(self):
        pred_classes = []
        for row in range(self.k):
            pred_classes.append(self.train_set[self.nearest_neighbors[row]][0])
        ### Referenced NumPy unique to return the predicted class
https://numpy.org/doc/stable/reference/generated/numpy.unique.html
        classes, counts = np.unique(pred_classes, return_counts=True)
        if len(counts) == 1:
            return classes
        if counts[0] > counts[1]:
            return classes[0]
        return classes[1]
```

```
    def train(self):
        ### Test every testing entry to training entry
        nearest_neighbors = []
        for train_row in self.train_set:
            nearest_neighbors.append(self.distance(train_row[1:] - self.test_set[1:]))
        ### Referenced NumPy argsort to sort by index of distances
https://numpy.org/doc/stable/reference/generated/numpy.argsort.html
        self.nearest_neighbors = np.argsort(nearest_neighbors)

class Feature_Selection:
    def __init__(self, dataset=None, k=None):
        self.dataset=dataset
        self.dataset_backwards=copy.deepcopy(self.dataset)
        self.k=k
        self.feature_set = []
        self.feature_set_backwards = []
        self.feature_subset = pd.DataFrame(dataset['label'])
        self.feature_subset_backwards = copy.deepcopy(self.dataset)
        self.best_features = [-1, -1]

    def print_best_features(self):
        print("Finished search!!! The best feature subset is {" +
self.best_features[0] + "}, which has an accuracy of " +
str(self.best_features[1]*100) + "%")

    def greedy_forward(self):
        if self.dataset.shape[1] == 1:
            return
        precision = []
        feature_name = []
        ### Retreiving the Columns for Feature Selection
        for column in self.dataset.columns:
            if column == 'label':
                continue
            feature = pd.concat([self.feature_subset,
pd.DataFrame(self.dataset[column])], axis=1).to_numpy()
            correct = 0
            ### K-Fold CV
            for cfv in range(int(len(feature)/self.k)):
                train_data = np.delete(feature, cfv, axis=0)
                test_data = feature[cfv]

                knn = KNN_Classifier(train_data, test_data, 1)
                knn.train()
                if knn.pred_class()[0] == test_data[0]:
                    correct += 1
            precision.append(correct/int(len(feature)/self.k))
            feature_name.append(column)

            if self.feature_set:
                feat_str = ','.join(str(feature) for feature in self.feature_set)
                print("\tUsing feature(s) {" + feat_str + "," + str(column) + "}
```

```
accuracy is " + str(correct/int(len(feature)/self.k)*100) + '%')
            else:
                print("\tUsing feature(s) {" + str(column) + "} accuracy is " +
str(correct/int(len(feature)/self.k)*100) + '%')

        precision_index = np.argsort(precision)
        self.feature_set.append(feature_name[precision_index[-1]])
        feat_str = ','.join(str(feature) for feature in self.feature_set)
        prev = self.best_features[1]
        self.best_features[1] = max(self.best_features[1],
precision[precision_index[-1]])
        if (self.best_features[1] != prev):
            self.best_features[0] = feat_str
        if precision[precision_index[-1]] < self.best_features[1]:
            print("\n(Warning, Accuracy has decreased!!! Continuing search in case of
local maxima)")
        print("\nFeature set {" + feat_str + "} was best, accuracy is " +
str(precision[precision_index[-1]]*100) + "%\n")
        self.feature_subset = pd.concat([self.feature_subset,
pd.DataFrame(self.dataset[feature_name[precision_index[-1]]])], axis=1)
        ### Referenced Pandas library to understand how to drop column
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html
        self.dataset = self.dataset.drop(columns=[feature_name[precision_index[-1]]])
        self.greedy_forward()
        pass

    def greedy_backward(self):
        if self.feature_subset_backwards.shape[1] == 1:
            return
        precision = []
        feature_name = []
        ### Retreiving the Columns for Feature Selection
        for column in self.dataset_backwards.columns:
            if column == 'label':
                continue
            feature = self.feature_subset_backwards.drop(columns=[column]).to_numpy()
            correct = 0
            ### K-Fold CV
            for cfv in range(int(len(feature)/self.k)):
                train_data = np.delete(feature, cfv, axis=0)
                test_data = feature[cfv]

                knn = KNN_Classifier(train_data, test_data, 1)
                knn.train()
                if knn.pred_class()[0] == test_data[0]:
                    correct += 1
            precision.append(correct/int(len(feature)/self.k))
            feature_name.append(column)

            if self.feature_set_backwards:
                feat_str = ','.join(str(feature) for feature in
self.feature_set_backwards)
```

```
                print("\tRemoving feature(s) {" + feat_str + "," + str(column) + "}
accuracy is " + str(correct/int(len(feature)/self.k)*100) + '%')
            else:
                print("\tRemoving feature(s) {" + str(column) + "} accuracy is " +
str(correct/int(len(feature)/self.k)*100) + '%')

        precision_index = np.argsort(precision)
        self.feature_set_backwards.append(feature_name[precision_index[-1]])
        feat_str = ','.join(str(feature) for feature in self.feature_set_backwards)
        prev = self.best_features[1]
        self.best_features[1] = max(self.best_features[1],
precision[precision_index[-1]])
        if (self.best_features[1] != prev):
            self.best_features[0] = feat_str
        if precision[precision_index[-1]] < self.best_features[1]:
            print("\n(Warning, Accuracy has decreased!!! Continuing search in case of
local maxima)")
        print("\nFeature set {" + feat_str + "} was best, accuracy is " +
str(precision[precision_index[-1]]*100) + "%\n")
        ### Referenced Pandas library to understand how to drop column
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html
        self.feature_subset_backwards = self.feature_subset_backwards.drop(columns=
[feature_name[precision_index[-1]]])
        self.dataset_backwards = self.dataset_backwards.drop(columns=
[feature_name[precision_index[-1]]])
        self.greedy_backward()
        pass


print("Welcome to KNN Feature Selection Algorithm!\n")
print("Please select the file you would like to test: ")
print("1. Sample Small 19\n" +
        "2. Assigned Small 21\n" +
        "3. Sample Large 6\n" +
        "4. Assigned Large 13\n")
file = input("Your selection: ")
file = int(file)
if file == 1:
        df = small_data_19
elif file == 2:
        df = small_data_21
elif file == 3:
        df = large_data_6
elif file == 4:
        df = large_data_13

# Asks the user the algorithm they would like to use
print("Please select the algorithm you would like to run: \n")
print("1. Forward Selection\n" +
        "2. Backward Elimination\n" +
        "3. Both\n")
algo = input("Your selection: ")
```

```
algo = int(algo)

print("This dataset has " + str(df.shape[1]-1) + " features (not including the class
attribute), with " + str(df.shape[0]) + " instances.\n")
correct = 0
feature = df.to_numpy()

if algo == 1:
        pred_classes = feature[:, 0]
        classes, counts = np.unique(pred_classes, return_counts=True)
        if counts[1] > counts[0]:
                print("Running KNN with no features, I get an accuracy of " +
str(counts[1]/(counts[0]+counts[1])*100) + "%\n")
        else:
                print("Running KNN with no features, I get an accuracy of " +
str(counts[0]/(counts[0]+counts[1])*100) + "%\n")

if algo == 2:
        for cfv in range(int(len(feature))):
                train_data = np.delete(feature, cfv, axis=0)
                test_data = feature[cfv]

                knn = KNN_Classifier(train_data, test_data, 1)
                knn.train()
                if knn.pred_class()[0] == test_data[0]:
                        correct += 1
        print("Running KNN with all " + str(df.shape[1]-1) + " features, using
\"leave-one-out\" evaluation, I get an accuracy of " + str(correct/len(feature)*100)+
"%\n")

feat_select = Feature_Selection(df, 1)
print("Beginning search.\n")
if algo == 1:
        start = time.time()
        feat_select.greedy_forward()
        feat_select.print_best_features()
        end = time.time()
        print("Elapsed (with compilation) = %s" % (end - start))
elif algo == 2:
        start = time.time()
        feat_select.greedy_backward()
        feat_select.print_best_features()
        end = time.time()
        print("Elapsed (after compilation) = %s" % (end - start))
elif algo == 3:
        start = time.time()
        print("Starting with Forward Selection.\n")
        feat_select.greedy_forward()
        feat_select.print_best_features()
        end = time.time()
        print("Elapsed (after compilation) = %s" % (end - start))
        start = time.time()
```

```
        print("Starting Backward Elimination.\n")
        feat_select.greedy_backward()
        feat_select.print_best_features()
        end = time.time()
        print("Elapsed (after compilation) = %s" % (end - start))
```

References

[1] Project 2 sample report file:
https://www.dropbox.com/scl/fo/kha82df4ml5zdf2tmvjij/ANjne70KBsct2yua9deRZOs?dl=0&e=1&preview=Project_2_sample_report_2024.docx&rlkey=jiyf5na8907o8ckkpqhtkha3j