Alejandro Sherman
SID: 862062898
No partner

CS170 Feature Selection With Nearest Neighbor Project Report

Due Sunday, June 13th

Data

My Specific Datasets

Dataset	Best Feature Set	Accuracy
Small Number: 92	Forward Selection = {2, 8}	0.98
	Backward Elimination = {2, 8}	0.98
Large Number: 92	Forward Selection = {6, 17}	0.967
	Backward Elimination = {6}	0.836

Sample Datasets

Dataset	Best Feature Set	Accuracy
Small Number: 80	Forward Selection = {5, 3}	0.92
	Backward Elimination = {3, 5}	0.92
Large Number: 80	Forward Selection = {27, 1}	0.951
	Backward Elimination = {27}	0.847

Full list of resources on 2nd Page.

Resource Links

In completing this project, I consulted following resources:

Part1:

Random Number Generation:

https://docs.python.org/3/library/random.html
http://www.easypythondocs.com/random.html
https://programming-idioms.org/idiom/70/use-clock-as-random-generator-seed/1087/python
https://pynative.com/python-random-seed/

Python List Operations:

https://www.studytonight.com/post/how-to-copy-elements-from-one-list-to-another-in-python
https://note.nkmk.me/en/python-list-clear-pop-remove-del/

Part2:

File Handling:

https://stackoverflow.com/questions/33048939/die-if-cant-open-file-in-python
https://docs.python.org/3/library/io.html#io.IOBase.readline
https://stackoverflow.com/questions/1904394/read-only-the-first-line-of-a-file

Data Handling:

https://stackoverflow.com/questions/44384854/split-string-into-array-in-python
https://stackoverflow.com/questions/32607370/python-how-to-get-the-number-of-lines-in-a-text-file

Normalization:

https://en.wikipedia.org/wiki/Feature_scaling
Lecture slides

Number formatting:

https://stackoverflow.com/questions/61834897/round-down-a-number-to-1-decimal-place
https://stackoverflow.com/questions/7604966/maximum-and-minimum-values-for-ints

Euclidean Distance:

https://en.wikipedia.org/wiki/Euclidean_distance
Lecture slides

Nearest Neighbor:

https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
Lecture slides

Leave One Out:

Lecture slides

Introduction

The feature selection project aims to be able to act as something of an example machine learning harness in which, when given a dataset consisting of many instances that contain one of two classes to be specified (1 and 2), and a number features to go along with that specific instance. The aim of the feature selection project is for the program to reliably find the most accurate features to include when determining the class of an unknown instance. Along with finding the best features to include, the project should also be able to find the accuracy, or the percentage of times that a certain combination of features leads to the correct guess of a classification. In this way, this project also shows how such a comparison compares with the default rate of predictions. The default rate being the pure statistic of the number of the most common class, over the total number of instances. Furthermore, this feature selection project aims to depict the difference in greedy selection algorithms when it comes to choosing the feature combinations to test, these selection algorithms being forward selection and backwards elimination.

Design

Initial Choice

My implementation for project 2 is in python.
Much like with project 1, I feel that python allows me more flexibility when it comes to overall design and implementation. As with project 1, I'm very much still learning python as I use it, so there were many things that I looked up during the course of development, which are listed above.

Once again, I feel that my choice of python proved to be a good one as the project evolved through each part.

Additionally, my design for this project flowed nicely as I followed the new part structure implemented this quarter. This was great as, rather than taking on a lot at once, much like project 1, here I was more easily able to prioritize what to incrementally implement as I developed project 2 for each "checkpoint".

Part 1:

Specifically, when I began with just part 1 (search algorithms only), my main focus was first, to create the main driver code that the user will interact with. More than anything, I designed the driver to match the sample trace that was given specifically for part 1. This entailed getting the algorithm choice from the user, as well as the total number of features. At this point, there was no actual dataset provided, as part 1 would use a "random" evaluation function rather than actual nearest neighbor classifier, and leave one out validation.

Functions created in part 1(driver, random evaluation, forward selection, backwards elimination)

A word on the greedy search algorithms:
Forward selection starts with an empty feature set and works by adding one new feature at a time to the set of features, and selects which feature to add by greedily selecting the feature that results in the highest current accuracy.
Backwards elimination starts with a full feature set and works by removing one feature at a time, and selects which feature to remove by greedily selecting the removal that leads to the highest current accuracy. (A downside of backwards elimination, is that, unlike forward selection which will always consider features that haven't been added yet, at any stage of the search, once backward elimination removes a feature, it is always gone. Backwards elimination will only ever remove more features. This fact will be relevant as we examine the different results between the two algorithms later on.)

Part 1 Challenges:
The biggest challenge of part 1, was most certainly the implementation of the forward selection and backwards elimination search algorithms. (A special algorithm would come later if I were to make one) Truthfully, at first I wasn't sure how to start. However, once Rutuja gave a rundown of how the algorithms function (I was stuck on how the graph nodes were created in the slides) I understood what I could do. Ultimately, I went with an array/dict structure to hold the features as I populated them. Also, now that I understood what made these search algorithms greedy, I was able to build a model that passes in newer features in order, and chooses based on the random value that is returned. I first finished the implementation of the forward selection algorithm, and matched the trace output with the sample given. Working on backwards elimination afterwards was comparatively easy as the principle was the same, the difference being that we now remove features from the list rather than add them.
The second challenge of part 1 was funnily enough, making the random numbers and showing them in a nice format. I for a long time couldn't figure out the random number generation in python as I kept defaulting to the way it was done in C++. With the help of some resources (listed above) I was able to figure it out.

Part 2:
With my part 2 implementation, in keeping with the incremental design process encouraged by the part structure of this project, I opted to implement part 2 in it's own completely separate main file from part 1. In other words, the implementation for part 2 is wholly independent from part 1. That being said, I did utilize similar structures used in part 1 (driver code, function use etc). Part 2 would see me implementing the real evaluation function, one that utilizes the nearest neighbor classifier and leave one out

validation. (Note: I did not end up creating train and test methods. As Rutuja had mentioned in her piazza post, we aren't particularly "training" our model in this project. As such I simply passed in the row to leave out when performing leave one out directly in the function implementation.)
Functions created in part 2(driver, normalize, nearest neighbor, real evaluation)

Part 2 Challenges:
Part 2 was the first part to actually need to utilize data given to us. In particular, part 2 came with a sample small and large dataset along with a given set of features and their approximate corresponding accuracy, such that my evaluation function can be tested against. I must say that one of the major challenges from part 2, came from the need to read data in from a file, and then utilizing that data in a meaningful format. There are many resources online for how to do so in python, but the struggle mostly lay in choosing a method that would work for me. Ultimately I went with a method that read in the data and stored it in an array/list format. The first value data[i] corresponds to the instance number (Some of the first data I collected was the total number of features and number of instances) while data[i][j] contains the actual data of that row (with each feature iterable). The links used to help me do so are listed above. Actual implementation of the nearest neighbor and leave one out algorithms weren't so tough as nearest neighbor mostly consisted of implementing finding euclidean distance (which I am now familiar with since project 1) and implementing leave one out validation, which followed from the lecture explanation and is simpler than a k fold algorithm. Another challenge that arose during part 2 however was the need to normalize the data. I had initially gotten results that did not match with the sample accuracies given by the professor. I was stumped, I was pretty confident in my classifier and my validator, but was unsure where the issue was. Could it be the euclidean distance formula? No. After reading the help on piazza, I realized the issue laid in the fact that I was using min-max normalization rather than the far more common method for machine learning, z score normalization, or standardization. This feature scaling turns out to be more valuable for datasets that may contain outliers. With this change, my results were in an expected range to the sample values.

Part 3:
With parts 1 and 2 completed in tow, I looked to complete the full program. As Rutuja had mentioned previously, if we had implemented parts 1 and 2 nicely and fully, we should be able to trivally create the full project code. For the most part, this was indeed the case for me. After all, while both parts were designed to be standalone, each implementation utilized functions to perform specific actions. Functions that could be easily moved around.
New functions made for the final submission(default evaluation function)

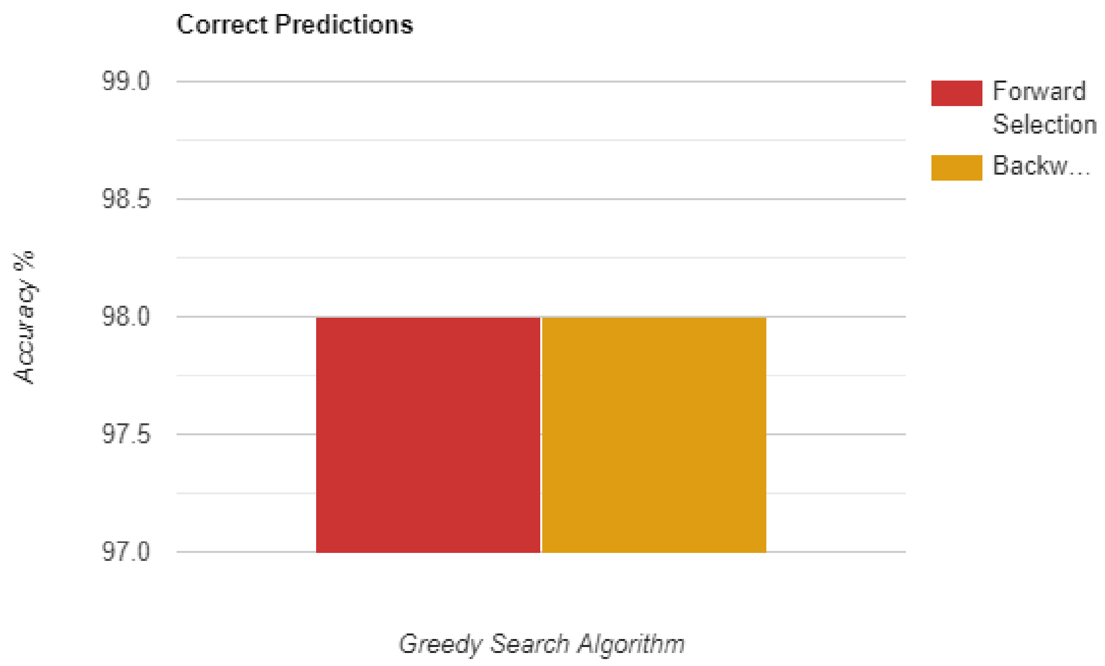
Part 3 Challenges:
As such, the biggest challenge of part 3 was deciding which program to start with as the basis of the full project, and then to move the contents of the other part into this program. I went with part 1 as the basis as it had a more fleshed out and similar driver function to what is required in the final trace. This meant that I then needed to carry over the file opening and data handling format from part 2 over myself. Additionally, I needed to replace my old "random" evaluation function with the real one from part 2, along with the nearest neighbor function. Here, most time was spent on matching the format of the data that the correct evaluation function took, as well as fitting everything together in an understandable format. Once this was done, I looked to update the format of the output to match the expected final trace, and as such I implemented a simple new function called "default_evaluation_function" made exclusively to find and output the default rate for forward selection before any features are selected. As Rutuja mentioned, this function simply finds the number of the most common class and outputs the percentage of that class in relation to all instances, acting as a prediction that is a bit better than a coin flip, as we saw in lecture. Similarly, for backwards selection, I would output an initial default rate that utilized every feature at once.

Performance & Data

Here I would like to list and compare the best features gained from each data set, their accuracies, and the differences between the results of forward and backwards selection. Two traces are given at the end of this report for forwards selection and backwards elimination on my unique given small dataset. (The rest of the traces are found in their own .txt files. [In the extra_traces folder] Additionally, I never ended up implementing a special algorithm.)

Unique Small Dataset (cs_170_small92.txt)
100 Instances, 10 Features

	Forward Selection:	Backwards Elimination:
Best Features:	[2, 8]	[2, 8]
Accuracy:		

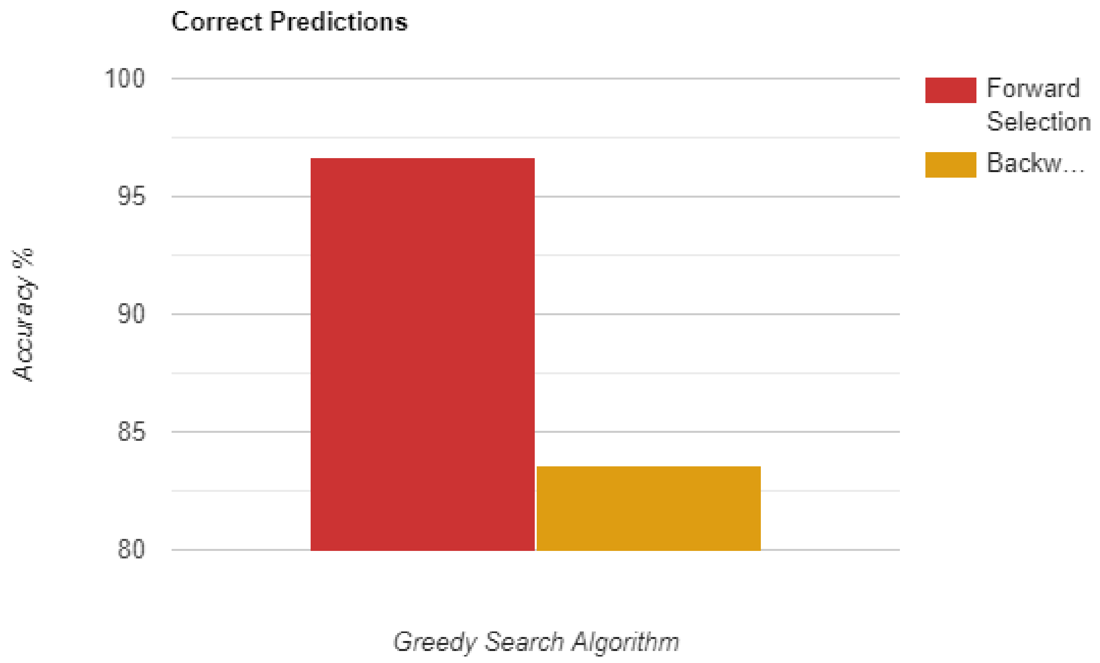


Here, each greedy search algorithm resulted with a combination of features 2 and 8. As such, both returned with the highest found accuracy of 98%.

Unique Large Dataset (cs_170_large92.txt)

1000 Instances, 40 Features

	Forward Selection:	Backwards Elimination:
Best Features:	[6, 17]	[6]
Accuracy:		



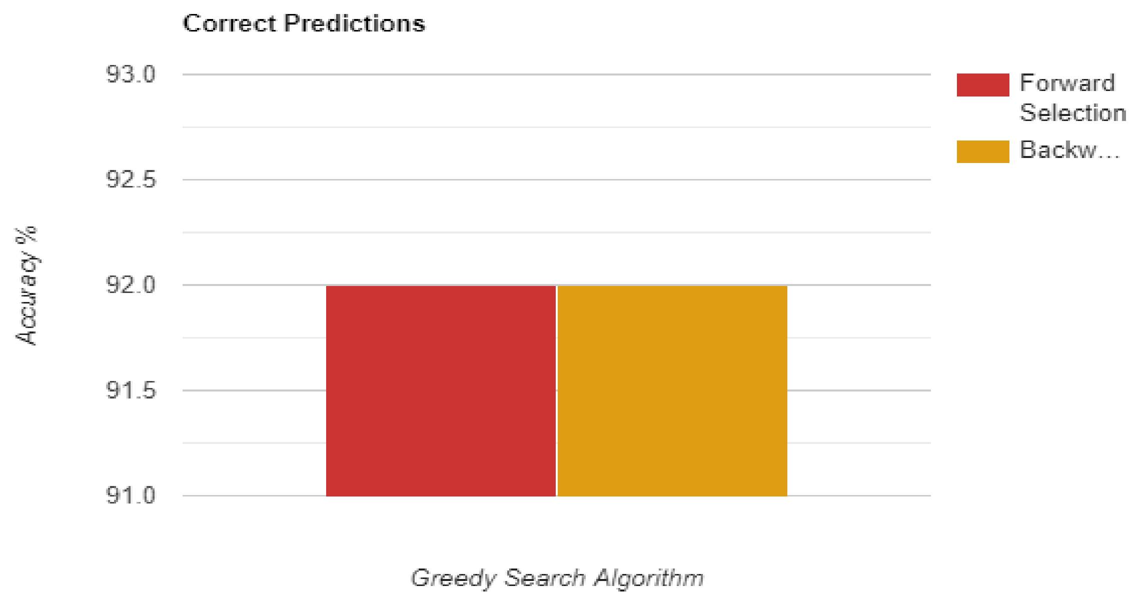
Here, we actually see each greedy algorithm come back with differing results. As mentioned before, backwards elimination can never recover a feature it had previously removed. In other words, backwards elimination will only ever remove. While forward selection on the other hand can never remove a feature once it has been added, it will always consider remaining unadded features at every step.

So in our case here, forward selection was able to simply choose the features 6, and 17 from the start and result with an accuracy of 96.7%. Backwards selection on the other hand chose to throw out feature 17 during execution as the values of the large combination of features seemed to bring down the overall accuracy. The result here then, was a large simply just one remaining feature of 6 and an end best accuracy of 83.6%. This due to the fact that the other "good" feature, 17, had already been removed, so all that remained was 6.

Now I will look into the similar data of the sample datasets given.

Sample Small Dataset (cs_170_small80.txt)
100 Instances, 10 Features

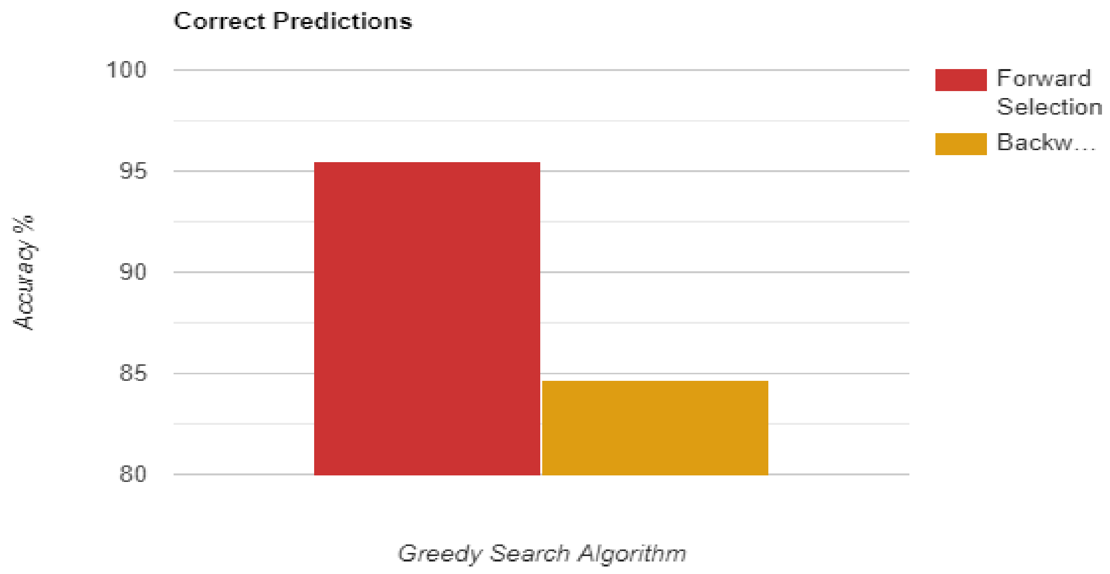
	Forward Selection:	Backwards Elimination:
Best Features:	[5, 3]	[3, 5]
Accuracy:		



Here, each greedy search algorithm resulted with a combination of features 3 and 5. As such, both returned with the highest found accuracy of 92%. The difference here being simply the order in which the features were found, forward selection finding feature 5 and then 3, while backwards selection always has features in numerical order.

Sample Large Dataset (cs_170_large80.txt)
1000 Instances, 40 Features

	Forward Selection:	Backwards Elimination:
Best Features:	[27, 1]	[27]
Accuracy:		



Again, we see each greedy algorithm come back with differing results for the large dataset case. Here, forward selection was able to simply choose the features 27, and 1 from the start and result with an accuracy of 95.5%. Backwards selection on the other hand chose to throw out feature 1 quite early on. The result here then, was a large simply just one remaining feature of 27 and an end best accuracy of 84.7%. This due to the fact that the other "good" feature, 1, had already been removed, so all that remained was 27.

A Word on Normalization

I mentioned before that my results didn't match the given sample accuracies during part 2 and the reason was due to using the incorrect normalization formula. Here I'd like to compare the results on my four datasets without normalization:

Unnormalized Unique Small Dataset (cs_170_small92.txt)

	Forward Selection:	Backwards Elimination:
Best Features:	[2, 8]	[2, 8]
Accuracy:	97%	97%

Here the features selected are the same as before. The accuracy however, is 1 percent lower than before, as the accuracy previously for these features was 98%. This is a minor change to be sure, but a difference nonetheless.

Unnormalized Unique Large Dataset (cs_170_large92.txt)

	Forward Selection:	Backwards Elimination:
Best Features:	[6, 17]	[6]
Accuracy:	96.7%	83.6%

Here the features and accuracies are unchanged from the values before. Here, lack of normalization has no visible effect.

Unnormalized Sample Small Dataset (cs_170_small80.txt)

	Forward Selection:	Backwards Elimination:
Best Features:	[5, 3]	[2, 4, 5, 7, 10]
Accuracy:	92%	83%

Here the result for forwards selection is just as before. With perhaps the most interesting result so far, backward elimination has a different set of values, namely [2, 4, 5, 7, 10]. Here, without normalization, backwards elimination seems to have thrown out a good feature, that being 3, and resulted with a lower accuracy of 83% by selecting a larger set of features to try to get the highest accuracy. In this case, we can see that normalization did in fact make a positive difference in the original data collection.

Unnormalized Sample Large Dataset (cs_170_large80.txt)

	Forward Selection:	Backwards Elimination:
Best Features:	[27, 1]	[27]
Accuracy:	95.5%	84.7%

Here the features and accuracies are unchanged from the values before. Here, lack of normalization has no visible effect.

Final Thoughts

Given the observations I have made above, there are a few things I feel I can comment on. Firstly, it's clear to me that, while often the same result, forward selection and backwards elimination occasionally will result in differently. This makes sense, as each follows a different greedy strategy, and both start at opposite ends of a spectrum, either with no features, or all of them and move from there. Generally, in my data, forward selection proved to be the more reliable and accurate of the algorithms. This is likely due to the type of datasets provided, and also the way in which my algorithms search. If the best number of features is about 2-4 of them, backwards elimination simply has much more work (work that may result in an otherwise good feature being discarded) to reach the end set with just a few features, while forward selection can be effectively done in a few steps as it finds the (presumed) highest accuracy relatively near the start.

Additionally, while not normalizing data may result in the same results of best features and best accuracy, leaving data as is, increases the chances that the algorithms will perform worse than they may have if they were normalized. While these variances tend to be minimal, and don't always happen, these differences are meaningful and real. After all, especially when using a nearest neighbor classifier, the data can be prone to being affected by a lack of normalization. As such, it's clear the value of normalization/standardization is great when it comes to feature selection in machine learning, even if it's not always immediately obvious.

As a final word, I'd like to mention that calculations for the larger datasets took quite a while in my model. (Hours) So I'd like to think that to further increase computational speed of the greedy algorithms, sorting and pruning of the data would be methods able to make improvements further than what was seen in my results.

Small Traces (also included in their own .txt files)

Forward selection trace for my unique small dataset (cs_170_small92.txt):

```
Welcome to Alejandro Sherman's Feature Selection Algorithm.

Type the name of the file to test: cs_170_small92.txt
Type the number of the algorithm you want to run.
    1    Forward Selection
    2    Backwards Elimination
    3    Alejandro's Special Algorithm

1

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

Please wait while I normalize the data...
Done!


Running nearest neighbor with no features (default rate), using "leaving-one-out"
evaluation, I get an accuracy of 83.0%

Beginning search.
```

```
Using feature(s) [1] accuracy is 75.0%
Using feature(s) [2] accuracy is 85.0%
Using feature(s) [3] accuracy is 67.0%
Using feature(s) [4] accuracy is 73.0%
Using feature(s) [5] accuracy is 65.0%
Using feature(s) [6] accuracy is 73.0%
Using feature(s) [7] accuracy is 76.0%
Using feature(s) [8] accuracy is 75.0%
Using feature(s) [9] accuracy is 77.0%
Using feature(s) [10] accuracy is 76.0%

Feature set [2] was best, accuracy is 85.0%

Using feature(s) [2, 1] accuracy is 84.0%
Using feature(s) [2, 3] accuracy is 78.0%
Using feature(s) [2, 4] accuracy is 85.0%
Using feature(s) [2, 5] accuracy is 90.0%
Using feature(s) [2, 6] accuracy is 83.0%
Using feature(s) [2, 7] accuracy is 87.0%
Using feature(s) [2, 8] accuracy is 98.0%
Using feature(s) [2, 9] accuracy is 81.0%
Using feature(s) [2, 10] accuracy is 90.0%

Feature set [2, 8] was best, accuracy is 98.0%

Using feature(s) [2, 8, 1] accuracy is 95.0%
Using feature(s) [2, 8, 3] accuracy is 90.0%
Using feature(s) [2, 8, 4] accuracy is 88.0%
Using feature(s) [2, 8, 5] accuracy is 90.0%
Using feature(s) [2, 8, 6] accuracy is 87.0%
Using feature(s) [2, 8, 7] accuracy is 91.0%
Using feature(s) [2, 8, 9] accuracy is 89.0%
Using feature(s) [2, 8, 10] accuracy is 92.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 8, 1] was best, accuracy is 95.0%

Using feature(s) [2, 8, 1, 3] accuracy is 85.0%
Using feature(s) [2, 8, 1, 4] accuracy is 87.0%
Using feature(s) [2, 8, 1, 5] accuracy is 84.0%
Using feature(s) [2, 8, 1, 6] accuracy is 76.0%
Using feature(s) [2, 8, 1, 7] accuracy is 87.0%
Using feature(s) [2, 8, 1, 9] accuracy is 79.0%
Using feature(s) [2, 8, 1, 10] accuracy is 89.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 8, 1, 10] was best, accuracy is 89.0%

Using feature(s) [2, 8, 1, 10, 3] accuracy is 88.0%
Using feature(s) [2, 8, 1, 10, 4] accuracy is 87.0%
Using feature(s) [2, 8, 1, 10, 5] accuracy is 85.0%
Using feature(s) [2, 8, 1, 10, 6] accuracy is 85.0%
Using feature(s) [2, 8, 1, 10, 7] accuracy is 81.0%
Using feature(s) [2, 8, 1, 10, 9] accuracy is 85.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 8, 1, 10, 3] was best, accuracy is 88.0%

Using feature(s) [2, 8, 1, 10, 3, 4] accuracy is 77.0%
Using feature(s) [2, 8, 1, 10, 3, 5] accuracy is 79.0%
Using feature(s) [2, 8, 1, 10, 3, 6] accuracy is 76.0%
Using feature(s) [2, 8, 1, 10, 3, 7] accuracy is 80.0%
Using feature(s) [2, 8, 1, 10, 3, 9] accuracy is 76.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 8, 1, 10, 3, 7] was best, accuracy is 80.0%

Using feature(s) [2, 8, 1, 10, 3, 7, 4] accuracy is 71.0%
Using feature(s) [2, 8, 1, 10, 3, 7, 5] accuracy is 80.0%
Using feature(s) [2, 8, 1, 10, 3, 7, 6] accuracy is 76.0%
Using feature(s) [2, 8, 1, 10, 3, 7, 9] accuracy is 71.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 8, 1, 10, 3, 7, 5] was best, accuracy is 80.0%

Using feature(s) [2, 8, 1, 10, 3, 7, 5, 4] accuracy is 71.0%
Using feature(s) [2, 8, 1, 10, 3, 7, 5, 6] accuracy is 81.0%
Using feature(s) [2, 8, 1, 10, 3, 7, 5, 9] accuracy is 63.0%
```

```
(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 8, 1, 10, 3, 7, 5, 6] was best, accuracy is 81.0%

Using feature(s) [2, 8, 1, 10, 3, 7, 5, 6, 4] accuracy is 75.0%
Using feature(s) [2, 8, 1, 10, 3, 7, 5, 6, 9] accuracy is 71.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 8, 1, 10, 3, 7, 5, 6, 4] was best, accuracy is 75.0%

Using feature(s) [2, 8, 1, 10, 3, 7, 5, 6, 4, 9] accuracy is 71.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 8, 1, 10, 3, 7, 5, 6, 4, 9] was best, accuracy is 71.0%

Finished search!! The best feature subset is [2, 8], which has an accuracy of 98.0%
```

Backwards elimination trace for my unique small dataset (cs_170_small92.txt):

```
Welcome to Alejandro Sherman's Feature Selection Algorithm.

Type the name of the file to test: cs_170_small92.txt
Type the number of the algorithm you want to run.
        1      Forward Selection
        2      Backwards Elimination
        3      Alejandro's Special Algorithm

2

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

Please wait while I normalize the data...
Done!


Running nearest neighbor with all features, using "leaving-one-out" evaluation, I get an
accuracy of 86.0%
```

```
Beginning search.

Using feature(s) [2, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is 78.0%
Using feature(s) [1, 3, 4, 5, 6, 7, 8, 9, 10] accuracy is 73.0%
Using feature(s) [1, 2, 4, 5, 6, 7, 8, 9, 10] accuracy is 73.0%
Using feature(s) [1, 2, 3, 5, 6, 7, 8, 9, 10] accuracy is 71.0%
Using feature(s) [1, 2, 3, 4, 6, 7, 8, 9, 10] accuracy is 71.0%
Using feature(s) [1, 2, 3, 4, 5, 7, 8, 9, 10] accuracy is 72.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 8, 9, 10] accuracy is 76.0%
Using feature(s) [1, 2, 3, 4, 5, 6, 7, 9, 10] accuracy is 68.0%
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 77.0%

Feature set [2, 3, 4, 5, 6, 7, 8, 9, 10] was best, accuracy is 78.0%

Using feature(s) [3, 4, 5, 6, 7, 8, 9, 10] accuracy is 73.0%
Using feature(s) [2, 4, 5, 6, 7, 8, 9, 10] accuracy is 79.0%
Using feature(s) [2, 3, 5, 6, 7, 8, 9, 10] accuracy is 79.0%
Using feature(s) [2, 3, 4, 6, 7, 8, 9, 10] accuracy is 77.0%
Using feature(s) [2, 3, 4, 5, 7, 8, 9, 10] accuracy is 67.0%
Using feature(s) [2, 3, 4, 5, 6, 8, 9, 10] accuracy is 78.0%
Using feature(s) [2, 3, 4, 5, 6, 7, 9, 10] accuracy is 71.0%
Using feature(s) [2, 3, 4, 5, 6, 7, 8, 10] accuracy is 76.0%
Using feature(s) [2, 3, 4, 5, 6, 7, 8, 9] accuracy is 77.0%

Feature set [2, 4, 5, 6, 7, 8, 9, 10] was best, accuracy is 79.0%

Using feature(s) [4, 5, 6, 7, 8, 9, 10] accuracy is 78.0%
Using feature(s) [2, 5, 6, 7, 8, 9, 10] accuracy is 77.0%
Using feature(s) [2, 4, 6, 7, 8, 9, 10] accuracy is 75.0%
Using feature(s) [2, 4, 5, 7, 8, 9, 10] accuracy is 75.0%
Using feature(s) [2, 4, 5, 6, 8, 9, 10] accuracy is 81.0%
Using feature(s) [2, 4, 5, 6, 7, 9, 10] accuracy is 72.0%
Using feature(s) [2, 4, 5, 6, 7, 8, 10] accuracy is 74.0%
Using feature(s) [2, 4, 5, 6, 7, 8, 9] accuracy is 74.0%

Feature set [2, 4, 5, 6, 8, 9, 10] was best, accuracy is 81.0%

Using feature(s) [4, 5, 6, 8, 9, 10] accuracy is 81.0%
Using feature(s) [2, 5, 6, 8, 9, 10] accuracy is 83.0%
Using feature(s) [2, 4, 6, 8, 9, 10] accuracy is 78.0%
Using feature(s) [2, 4, 5, 8, 9, 10] accuracy is 79.0%
Using feature(s) [2, 4, 5, 6, 9, 10] accuracy is 75.0%
Using feature(s) [2, 4, 5, 6, 8, 10] accuracy is 78.0%
Using feature(s) [2, 4, 5, 6, 8, 9] accuracy is 84.0%

Feature set [2, 4, 5, 6, 8, 9] was best, accuracy is 84.0%

Using feature(s) [4, 5, 6, 8, 9] accuracy is 79.0%
Using feature(s) [2, 5, 6, 8, 9] accuracy is 87.0%
Using feature(s) [2, 4, 6, 8, 9] accuracy is 82.0%
Using feature(s) [2, 4, 5, 8, 9] accuracy is 79.0%
Using feature(s) [2, 4, 5, 6, 9] accuracy is 76.0%
Using feature(s) [2, 4, 5, 6, 8] accuracy is 81.0%

Feature set [2, 5, 6, 8, 9] was best, accuracy is 87.0%

Using feature(s) [5, 6, 8, 9] accuracy is 82.0%
Using feature(s) [2, 6, 8, 9] accuracy is 84.0%
Using feature(s) [2, 5, 8, 9] accuracy is 84.0%
Using feature(s) [2, 5, 6, 9] accuracy is 81.0%
Using feature(s) [2, 5, 6, 8] accuracy is 87.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2, 5, 6, 8] was best, accuracy is 87.0%

Using feature(s) [5, 6, 8] accuracy is 72.0%
Using feature(s) [2, 6, 8] accuracy is 87.0%
Using feature(s) [2, 5, 8] accuracy is 90.0%
Using feature(s) [2, 5, 6] accuracy is 79.0%

Feature set [2, 5, 8] was best, accuracy is 90.0%

Using feature(s) [5, 8] accuracy is 78.0%
Using feature(s) [2, 8] accuracy is 98.0%
Using feature(s) [2, 5] accuracy is 90.0%

Feature set [2, 8] was best, accuracy is 98.0%
```

```
Using feature(s) [8] accuracy is 75.0%
Using feature(s) [2] accuracy is 85.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [2] was best, accuracy is 85.0%

Using feature(s) [] accuracy is 83.0%

(Warning, Accuracy has decreased! Continuing search in case of local maxima)
Feature set [] was best, accuracy is 83.0%

Finished search!! The best feature subset is [2, 8], which has an accuracy of 98.0%
```