Justin Pham

SID: 862114467

Email: [jpham079@ucr.edu](mailto:jpham079@ucr.edu)

Dec 10, 2021

Project 2 Machine Learning with Dr. Eamonn Keogh

My code is heavily based on the MatLab code provided by Dr. Keogh. All code is original except for functions from following Python libraries.

* **csv**, **os**, **pandas** and **numpy** were used to parse the provided text file
* **sys** and **math** were used to perform cross validation and calculate Euclidean distance respectively
* **time** was used to record how long the program took to run

I also used the Cython compiler to speed up my search. Cython is a tool that converts Python code into C code. I used the template from this website to compile my Python code: <https://neurohackademy.github.io/high-performance-python/03-compiling/>. The code is compiled by running “python source/compile.py build\_ext --inplace” from the root directory. Run the code with “python run.py” from the root directory..

Link to my code: <https://github.com/Pjsrcool/CS-170-Project-2>

**Premise**

Dr. Keogh assigned each student a small data set and a large data set. Our task is to use Forward Selection and Backward Elimination to find the most relevant features in the data sets.

**Results for Small Data**

I was assigned Ver\_2\_CS170\_Fall\_2021\_Small\_data\_\_91.txt.

In Figure 1 below, we see the results of running Forward Selection on the Small data. For the empty set [ ], I reported the Default rate, which is 84.0%. The Greedy Search started by selecting [6] with an accuracy of 86.2%. The next iteration finds features [6,2], then it finds [6,2,5], and so on. After all the search iterations, we find that the features [6,2] would be the most accurate at 97.4 % accuracy. I believe it may be over-fit. Therefore, I believe features [6,2,5] to be the best features, with accuracy of 95.8 %. It is only 1.6% less accurate, but has more features. This means that it maintains basically the same accuracy while using more features, so there is less chance if it being an over-fit.

In Figure 2 below, we see the results of running Backward Elimination on the Small data. I started by reporting the accuracy using All Features, which is 79.0%. Then after removing Feature 1, we have an improved accuracy of 81.2%. In the next iteration, Feature 9 is removed for an improved accuracy of 83.2%. After all the iterations complete, I find that we have the same results as Forward Selection, which is features [2,5,6] is the best answer with 95.8%, and that features [2,6] is most likely and over-fit with an accuracy of 97.4%.

**Conclusion for Small Data:**

My final answer for the Small Data set is that features [2,5,6] is the best. Using features [2,6] may result in an over-fitting because it is marginally more accurate, but uses less features. It may make a really good model for the data we have, but it may also perform worse when deployed in the real world.

**Results for LARGE Data**

I was assigned Ver\_2\_CS170\_Fall\_2021\_LARGE\_data\_\_91.txt.

In Figure 3 below, we see the results of running Forward Selection on the LARGE Data. For the empty set, I reported the Default Rate which is 84.4% accuracy. Then I added Feature 34 for a marginal decrease to 84.2% accuracy. After that, adding Feature 29 for a set of [34,29] results in a large increase to 97.2% accuracy. This is the highest accuracy recorded. After all the iterations, I believe the best answer from Forward Selection is features [34,29,20] for an accuracy of 95.7%. I believe this because it is only 1.5% less accurate than features [34,29]. Similar to the Small Data set, this is most likely an issue with overfitting, having 3 features with minimally lower accuracy reduces the chances of overfitting.

In Figure 4 below, we see the results of running Backward Elimination on the LARGE Data. At the start, I report the accuracy of all the features, which is 73.0%. In the first iteration, I removed Feature 29 for an improved accuracy of 74.1%. In the next iteration, I removed Feature 34 for an improved accuracy of 74.6%. After all the iterations, I somehow find the the best set of features would be the set of no features [ ], for an accuracy of 84.4%. That means the most accurate model is a random guess. This is interesting because we found in the Forward Selection that there exists sets of features with high accuracy. I believe we have this result because we search through the features using a greedy search, and that the set [ ] is a local maxima. For example, maybe we removed Feature 11 at level 29 because it slightly increases our accuracy. But we might get better results if we remove a different feature the decreases our accuracy temporarily, only to significantly increase our accuracy later on. Basically using greedy search for the features guarantees that we get stuck at a local maxima.

**Conclusion for LARGE Data:**

Since Backward Elimination returned an obviously suboptimal answer, I can only take the result from Forward Selection as my correct answer. To summarize, I believe features [34,29,20] to be the best answer because it is the perfect compromise between overfitting and accuracy. By having 3 features instead of two, we are more likely to have a strong correlation. By sacrificing 1.5% inaccuracy, we reduce the chances of our model being overfit.

**Computational Effort for Search**

The search was performed on a laptop with an Intel i7 9750H with 16 GB of RAM. Table 1 shows how long it took for the laptop to perform the searches.

|  |  |  |
| --- | --- | --- |
|  | Table 1 |  |
|  | Small Data Set | LARGE Data Set |
| Forward Selection | 5.2 Seconds | 2.2 Hours |
| Backward Elimination | 6.6 Seconds | 3.6 Hours |

I originally used only Python, but it was taking hours to run. I was able to use Cython to compile my Python code into C code, which improved the runtime about 30%. For example, the LARGE Data set originally took 3.5 hours to compute, but now computes in 2.2 hours with Cypthon.

Below is a trace of the Small Data 91 that was assigned to me. For this trace, I used Forward Selection only.

Enter 1 for small data set.

Enter 2 for large data set.

--> 1

Enter 1 for Forward Selection.

Enter 2 for Backward Elimination.

--> 1

the default rate is (empty set) is 0.8396793587174348

on the 1th level of search tree

consider adding the feature 1

consider adding the feature 2

consider adding the feature 3

consider adding the feature 4

consider adding the feature 5

consider adding the feature 6

consider adding the feature 7

consider adding the feature 8

consider adding the feature 9

consider adding the feature 10

on level 1 we added features 6

[6] accuracy: 0.8617234468937875

// I removed the trace for levels 2-9 to save space

// Shown are the searches for level 1, level 10, and the final result

on the 10th level of search tree

consider adding the feature 1

on level 10 we added features 1

[6, 2, 5, 8, 4, 9, 7, 3, 10, 1] accuracy: 0.7895791583166333

best set is: [6, 2] with accuracy: 0.9739478957915831

[] --> accuracy 0.8396793587174348

[6] --> accuracy 0.8617234468937875

[6, 2] --> accuracy 0.9739478957915831

[6, 2, 5] --> accuracy 0.9579158316633266

[6, 2, 5, 8] --> accuracy 0.9238476953907816

[6, 2, 5, 8, 4] --> accuracy 0.8857715430861723

[6, 2, 5, 8, 4, 9] --> accuracy 0.8537074148296593

[6, 2, 5, 8, 4, 9, 7] --> accuracy 0.8597194388777555

[6, 2, 5, 8, 4, 9, 7, 3] --> accuracy 0.8296593186372746

[6, 2, 5, 8, 4, 9, 7, 3, 10] --> accuracy 0.811623246492986

[6, 2, 5, 8, 4, 9, 7, 3, 10, 1] --> accuracy 0.7895791583166333

runtime: 5.1565961837768555 seconds