CS181 Practical 1 - Writeup

Victor Domene
victordomene@college.harvard.edu
Henrique Vaz
hvaz@college.harvard.edu
Stefan Gramatovici
sgramatovici@college.harvard.edu

February 11th, 2016

Contents

1 Overview

In this practical, we had the task to predict the HUMO-LUMO gaps of a dataset of over 800,000 molecules (as given by their smile serialization), given the gaps for another 1,000,000 molecules. We were given a simple script that performed naive Linear Regression and Random Forest Regression (without any parameter tuning).

In order to accomplish this open-ended problem, we had to figure out exactly what the best approach would be. We were given only 256 features (of origin unknown) to the train and test data, and perhaps reducing or increasing that number could be beneficial to the predictions. Indeed, several of those columns were unused across all the 1,000,000 training molecules. On the other hand, the methods used in the given code were very simple and could be tuned (for instance, increasing the number of random decision trees, or using a Lasso or Ridge regression rather than a plain linear one). We could even use more complex algorithms, such as Gaussian Processes or Neural Networks.

Our team decided that we wanted to explore all of these areas. To start, we used the Python RDKit module in order to produce new features to our data. Then, we chose a few models we wanted to try out. Finally, we patiently waited for Python and all its slowness to finish running our scripts (and really, this was probably the worst and most time-consuming part of this assignment).

In this writeup, we will provide some details on how we proceeded on each step of this assignment.

2 Feature Engineering

At first, we thought that the features provided would be enough to run our methods and get reasonable results. However, after a simlpe check on the variance of the columns, we got a staggering result: only 31 columns had non-zero variance. So, most of the provided data was garbage, and was not helping our regression in any way (other than, possibly, slowing it down).

Given these circumstances, we had to explore the richness of the RDKit package. We started by adding simple features such as number of atoms, number of bonds and other things that we thought could be influential in the resulting gap. However, these did not seem to improve our methods very much. After some more research, we found out about fingerprints, which are widely used in machine learning (according to the RDKit's own website).

Due to the lack of specific chemical knowledge, we decided to use the popular Morgan Fingerprinting scheme for building features. Due to memory limitations on our computers, we initially built fingerprints with bitvectors of size 128, 256, 512 and 1024. We built these datasets using RDKit. All of our benchmarks were made on top of these features, which we considered to be much more informative than the given data.

3 Method Selection

Among all the possible methods that we could try, we chose some methods that we thought would be feasible, efficient and interesting. So, we went a little bit out of our way to understand

some new algorithms that we had never seen before. We did use Python libraries for the methods themselves, but we studied each method's advantages and limitations before running them (more on the next section of this writeup).

Given the constraints on our machines, we decided to compare the methods on smaller datasets. For this purpose, we sampled 5000, 10000, 20000, 50000 and 100,000 molecules. These allowed us to calculate cross validation scores according to the RMSE. We used 3-fold validation, in most of the cases. We ran all of our tests against the 5000, 10000 and 20000 datasets. We decided however to only focus on Random Forests and Neural Networks for the larger datasets (and eventually, the full 1 million molecules dataset). We provide a table of results on our smaller datasets below, where all the information was obtained by training on the 256 feature version of the Morgan Fingerprint dataset (for some consistency with the given data from the assignment). We also ran the same regressions on 128 and 512 features, but the conclusions on which methods worked best were the same. For the sake of concreteness and clarity, we will save the reader from huge tables and provide only the 256 version.

3.1 Ridge Regression

This was our firsti real idea: use a simple Ridge Regression. We just wanted to try this out and see if it would differ from the regular Linear Regression model provided. We used the implementation from `sklearn.linear_models`).

# of molecules	RMSE
5000	0.214717311039
10000	0.208822217082
20000	0.208394415786

The positive results here came up as a surprise. This worked fairly well in our cross validated data, even for larger datasets. The high-dimensionality of this problem and the fact that Ridge tends to avoid overfitting could help explain why this would be a decent fit.

3.2 Lasso Regression

Similarly to our motivation for Ridge Regression, we thought Lasso Regression could be useful in this analysis. Again, we used the implementation given inm `sklearn.linear_models`. We used three parameters for α, the weight on the L1 norm: 0.5, 1.0 and 1.5.

# of molecules	α	RMSE
5000	0.5	0.419778474976
5000	1.0	0.419778474976
5000	1.5	0.419778474976
10000	0.5	0.419733018992
10000	1.0	0.419733018992
10000	1.5	0.419733018992
20000	0.5	0.419733000248
20000	1.0	0.419733000248
20000	1.5	0.419733000248

This was one of the most disappointing results. At first, we had no idea that Lasso would have such bad performance. Thinking deeper into the issue and doing some research, it seems that minimizing the error with the L1 norm favors weights closer to 0, so that it works best when there are many features that are fairly irrelevant (as per this thread). In our fingerprints, this does not seem to be the case. Also, changing the values for α did not seem to affect the results.

3.3 Random Forests

From the regular models of Linear Regression, we decided that the results were not quite as satisfactory as one may desire. We moved on to exploring Random Forests, in the implementation of `sklearn.ensemble`. We did trials with 16, 32, 64 and 128 estimators.

# of molecules	# of estimators	RMSE
5000	16	0.19829303
5000	32	0.19460209
5000	64	0.19353467
5000	128	0.1913841
10000	16	0.17802491
10000	32	0.17509088
10000	64	0.17265379
10000	128	0.17126391
20000	16	0.16696353
20000	32	0.16380786
20000	64	0.16159417
20000	128	0.15943662

As seen from the plots, increasing the number of decision trees (estimators) decreases the error, but the marginal benefit is decreasing. We have thus considered that 64 estimators would be good enough for our purposes, and also computationally feasible. Also, the error seems to decrease as we insert more data, which was promising for usage in the entire dataset.

3.4 AdaBoost

In the spirit of attempting to explore new algorithms, we decided to try AdaBoostRegressor, present also in `sklearn.ensemble`. We were hoping this would give incredible results, since it is an Ensemble method (which seemed promising from the Random Forest data) which is "boosted". Our expectations, however, were not met. In all of our tests, AdaBoost performed incredibly poorly when compared to Random Forests, even when the number of estimators was the same. We tested this for several values for the learning rate, and all of them produced similar results. Perhaps it was a lack of knowledge of how to tune it further, but this generally did not seem to work.

Since AdaBoost was reasonably fast, we did a lot of testing on it. We will only show some of these results in this report.

# of molecules	# of estimators	Learning Rate	RMSE
5000	64	0.7	0.23539529
5000	64	1.0	0.23024093
5000	64	1.5	0.23167797
10000	128	0.7	0.23432945
10000	128	1.0	0.23044077
10000	128	1.5	0.22898739
20000	128	0.5	0.23316869
20000	128	1.0	0.23283003
20000	128	1.5	0.22835169

Even though increasing the number of estimators does decrease the error, this change is not as significant as the Random Forest. In general, this method did not seem to work very well for our data. We could not exactly predict the reason for this. In general, it seemed like increasing α slightly improved the score, but this was not too significant (in a very informal definition of significant).

3.5 Neural Network

After being a bit disappointed with AdaBoost, we wanted to explore a new type of algorithm. Even though there were several types of different algorithms for implementing Neural Networks, most of which would involve libraries such as `Theano` or even Google's `TensorFlow`, we found an incredibly simple interface for Neural Networks: `PyBrain`. The simplicity of this library called our attention, and we thought it was worth a shot.

Without any pretentions, we ran the entire 1 million molecule dataset on a PyBrain network with only 3 hidden layer, using the 1024 features from Morgan Fingerprints and training for 5 epochs. For our surprise, not only did it run incredibly fast, but it also produced very good results. On Kaggle, this very simplistic approach got a RMSE of 0.18173, which actually was much better than the given samples.

4 Expectations and Challenges

We had no idea how long these training sessions would take. The first couple of attempts included running poorly written code directly on the 1 million dataset, which resulted in bugs after 4 hours of execution. After some time, however, we learned our lesson and started building from the smaller databases upwards.

As soon as we set the initial framework for producing features, running regressions and performing cross validation, the process was much faster (and we could run it over night). We were learning from the smaller datasets and improving our parameters after each iteration. This was a very challenging process (but fortunately, it did not stop us from gathering interesting information).

We were expecting the most complex algorithms to perform better. This, however, was not always true: AdaBoost was almost always off by approximately 0.2 when compared to Ridge Regression, for instance. This struck us as a surprise. But in the end, Random Forests and Neural Networks seemed to work the best, and we spent the last couple of days of the assignment focusing on those two.

5 Benchmarks on Selected Methods

From our results, we decided to run both Random Forests and Neural Networks on our 1 million molecules dataset. These were the ones that gave us the best scores on Kaggle (with Random Forest performing better). We also ran Ridge Regression (simply for comparison purposes).

5.1 Ridge Regression

This method was not only fast, but it also gave us nice results. With the entire dataset, Ridge Regression had a partial score of 0.14337 on Kaggle. This was suprisingly better than plain old-vanilla Linear Regression.

5.2 Random Forest

This was our best hope for a good score on Kaggle. Given the great results achieved by this method in the smaller datasets, we were hoping to get results at least as good as in those tests (which would already be better than the two benchmarks). We left a computer running the Random Forest for an entire night, and submitted it, getting a partial score of 0.05994, our best prediction. The program took about 6 hours to run to completion.

5.3 Neural Network

With the possibility of achieving even better results, we ran a new PyBrain neural network, but now for a maximum of 60 epochs. This training session took more than 12 hours to complete. Unfortunately, this did not give results as shocking as we expected. The RMSE score on Kaggle was 0.11129. This was still a good result, and may be even better when the entire dataset of tests in taken into account, when Kaggle closes the competition and final scores are released.

6 Code

The code we used for feature engineering, as well as for cross validation, score calculations and regressions can be found in the following GitHub Repository. It includes a README.md file with some descriptions on directories and dependencies.