

Practical 1: Regression

Predicting the Efficiency of Organic Photovoltaics

Antonio Copete (acopete@cfa.harvard.edu, Camelot.ai Copete)
Fangli Geng (fgeng@g.harvard.edu, Camelot.ai fangligeng)
Xihan Zhang (xihanzhang@hsph.harvard.edu, Camelot.ai Xihan)

February 10, 2018

1 Technical Approach

We began by performing exploratory analysis on the sample dataset we were given, which consisted of a training set of 1 million molecules with 256 binary predictors, in addition to their SMILES string and the HOMO–LUMO gap we were seeking to predict. The test set consisted of 824,230 molecules with the same set of predictors as well as their SMILES string. The sample code we were given implemented a default Linear Regression model on the full set of 256 predictors, yielding an $R^2_{\text{LR}} = 0.461$ ($\text{RMSE}_{\text{LR}} = 0.298$) over the full training set, and a default Random Forest regression with $R^2_{\text{RF}} = 0.554$ ($\text{RMSE}_{\text{RF}} = 0.272$).

Initial inspection found that out of 256 molecular features, 221 of them were unexpressed (i.e. had $x_i = 0$) for *all* molecules, both in the training set and the test set. Dropping unimportant features would normally call for K -fold cross-validation across the training set to ensure those features are consistently unimportant in all cases. However, in the case of null values of certain features for every element of the training set, it is not only legitimate but necessary to drop those features from all further analysis, as fitting along null dimensions would constitute a form of overfitting.

Having reduced the sample dataset to 31 expressed molecular features, we performed both regularized and non-regularized linear regression with cross-validation¹ and hyperparameter tuning, under the following methods:

1. Non-regularized linear regression: Yields a mean $R^2 = 0.461$ from 5-fold cross-validation, similar to the original result with 256 predictors.
2. Ridge Regression: Tuning the hyperparameter α by grid-search cross-validation results in an optimal value of 5.86×10^{-5} , with a best $R^2 = 0.459$, a result almost identical to non-regularized regression.
3. Elastic Net: Trying a more general regularization of the form $\alpha_1 L_1$ (Lasso) + $\alpha_2 L_2$ (Ridge regression), and tuning the L_1 ratio $\equiv \alpha_1/\alpha_2$ by grid-search cross validation, yields an optimal

¹After initially trying 3-fold, 5-fold and 10-fold cross-validation, we settled on 5-fold cross-validation as the best compromise between accuracy and computational speed.

value of 0.0, which indicates that that a regularization biased towards sparse models, such as the Lasso, might not be the best choice in this case.

The results from linear regression on the sample dataset consistently gave us relatively low predictive value, even when compared to generic random forest regression, which led us into 2 parallel tracks:

1. Feature engineering:

The RDKit package in Python use the SMILES string to provide an array of information and functions about a molecule, including substructure searching, chemical reaction, enumeration of molecular resonance structures, and several sorts of fingerprints. From our research we decided to extract the *Morgan fingerprint function*, which is widely used with SKLearn in ML and carries the information we considered to be most relevant to the efficiency of organic photovoltaics, such as: (a) Connectivity: Element, numbers of heavy neighbors, numbers of Hs, charge, isotope, inRing; (b) Chemical features: Donor, Acceptor, Aromatic, Halogen, Basic, Acidic; and (c) Taking into account the neighborhood of each atom.

The process takes the "SMILES" column from the training and test data and uses RDKit to convert it into molecule objects, followed by gathering of the Morgan fingerprint by use of the command `rdkit.Chem.getmorganfingerprintasbitvect()`, adjusting the parameter `nBits` to 512, 2048 and 4096 aiming to test datasets with various precision levels. After iterating this process for each string, we appended the columns together to form a new input matrix. The size of the dataset greatly varies with precision level, and for this reason we had to slice the datasets into 8 parts in order to accommodate computational constraints.

2. Non-linear methods:

Among these we concentrated on 2 broad categories:

(a) Tree-based Ensemble methods:

Given the binary predictors coincide with the branch-like structure of decision tree, we had a deeper dive into decision tree. Deep-depth tree may overfit the training set, so that ensemble methods were applied to take the average of predictions of all trees. The baseline ensemble methods are: (a) Bagging : bootstrap multiple training sets to fit multiple trees to reduce the variance. (b) Randomforest : further reduce the variance by randomly picking sub set of \sqrt{p} predictors from the total p predictors at each split to make. (c) Extremely Randomized Trees : further reduce the variance by selecting cutting point from sub set totally at random. (d) AdaBoost: use a linear combination of multiple simple trees to reduce the residual step by step.

Baseline models were fitted on the selected 31 features. 20% validation set was used to test MSE generated by each model. Bagging, Randomforest, Extremely Randomized Trees are quit similar to each other. Adaboost which focus on reducing bias rather than varriance, is much weaker. Thus, we selected random forest as main model, and tuned number of estimators(i.e. the total number of trees to take average on) to reduce variance of the model. 5-fold cross validation was used on the 1/7 of the total sample with 2048 newly generated features.

From result, 13 estimators is sufficient to make MSE of test set near training set. Given the total sample size is 7 times of the test set where we tuned the model, setting the number of estimators

to be 150 for the whole data set is safe and reasonable.

We also plot the distribution of importance of predictors based on the baseline random forest, the result is as below:

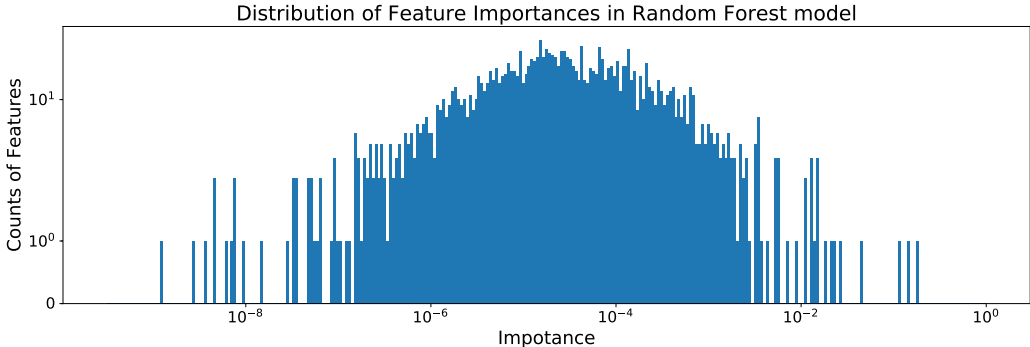


Figure 1: Histogram of feature importances from Random Forest

Most portion of predictors have little importance(10^{-8} to 10^{-2}), but combining them can yield good precision on validation set($R^2 = 0.936$, $MSE=0.0106$). This made us explore more on Deep learning method, which considers the interaction among features.

(b) Deep learning:

In order to model the interactions that would be expected among the broad set of molecular properties extracted from RDKit, we decided to explore the training of simple neural networks and got acquainted with the basics of Deep Learning as a tool for this problem. In all cases, we used an architecture of one or more dense (fully-connected) hidden layers between the input layer of N parameters and the output layer of 1 node for the response we wanted to model. For each node in the hidden layers, we used the widely used “relu” function $f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ as activation function. We trained the model using the *keras* module in Python, using the *Adam* optimizer for the learning rate, setting aside 20% of training data for validation. The model fits for the weights of the connections between pairs of nodes in the network, which in the case of 1 hidden layer with M nodes, would result in $(N + 1) \times M$ parameters to be trained.

Taking a trial-and-error approach to setting the network architecture, we show the results of validation set MSE for 3 broad cases, using the 31 expressed features in the sample training dataset: 1) 1 hidden layer with variable number of nodes, 2) 2 hidden layers, with 31 nodes on the first and variable number of nodes on the second, and 3) 3 hidden layers, with 31 nodes on each of the first two, and variable number of nodes on the third.

The result of this experiment showed that the MSE was not as sensitive to the addition of a

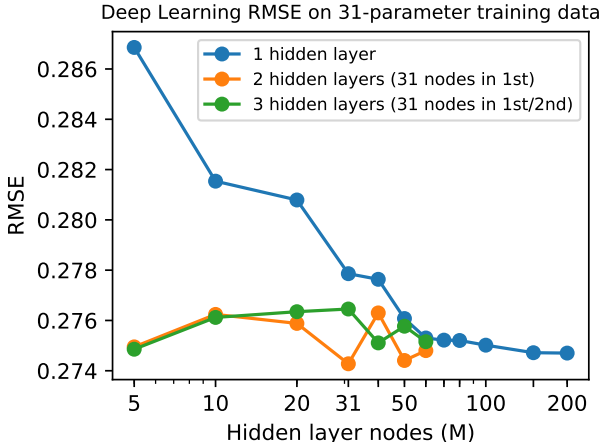


Figure 2: MSEs of Neural Network architectures

second and third layer, as it was to the number of nodes in the first layer of the 1-hidden layer architecture. Given the computational limitations that were expected for a much larger set of features, we used this result to use the feature-engineered set to train only 1-layer models with a limited number of nodes in the same range as the number of predictors.

2 Results

Figure 3 shows the overall results we obtained from a wide array of regression methods on different sets of features, as described in the Technical Approach section, and roughly listed in the order in which we tried them.

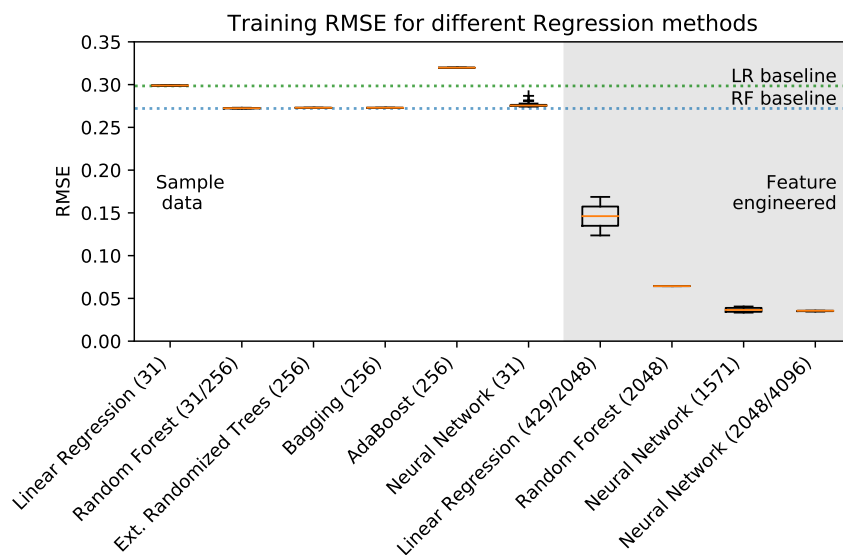


Figure 3: Overall results for different regression methods and number of predictors

One evident result is the marked improvement in the predictive value of feature-engineered models employing a large number of predictors (429/1571/2048/4096), compared to those from the sample dataset (31/256). Within models of comparable numbers of predictors, linear regression performed relatively poorly, indicating the non-linear interactions between features that are likely to result in the correct HOMO–LUMO gaps. Our 2 best results in the Camelot table came from employing neural network models, one with 4096 features and 1 hidden layer of X nodes (Fangli please fill in here), and another with 1571 features and 1 hidden layer of 500 nodes. These results ranked 9 (RMSE = 0.03461) and 11 (RMSE = 0.0359) in the final table, both in the top 10% of scores.

3 Discussion

End your report by discussing the thought process behind your analysis. This section does not need to be as technical as the others but should summarize why you took the approach that you did. Credit will be given for:

- *Explaining the your reasoning for why you sequentially chose to try the approaches you did (i.e. what was it about your initial approach that made you try the next change?).*
- *Explaining the results. Did the adaptations you tried improve the results? Why or why not? Did you do additional tests to determine if your reasoning was correct?*