Practical 1: Predicting the Efficiency of Organic Photovoltaics

Manav Khandelwal, Joel Beazer, Paul Stainier
manavkhandelwal@college.harvard.edu, beazer01@college.harvard.edu, pstainier@college.harvard.edu
manavkhandelwal, beazer01, MileyCyrus

February 9, 2018

1 Technical Approach

1.1 Feature Generation

1.1.1 RDKit Basic Features

Using RDKit, we were first able to generate some very basic features about the structures of the molecules in question. We chose these features because they both seemed relevant to the DFT calculations of the HOMO-LUMO gap, and were easy to extract using RDKit. The features we used are as follows: Total Atomic number, Average Atomic number, Total number of atoms, smallest set of smallest rings (SSSR). Taking inspiration from T1, we transformed each of the four variables using these two basis functions: $\phi_1(x) = x^j$ for $j = 2, 3, 4, 5, 6$, $\phi_2(x) = cos(\frac{x}{j})$ for $j = 1, 2, 3, 4, 5$. This led to a significant decrease in test RMSE from .27 to .16 using a random forest.

1.1.2 Fingerprint

The effectiveness of an organic photo-voltaic cell is determined by the HOMO-LUMO gaps in it's primary conductor material. These HOMO-LUMO gaps are influenced by several molecular features: conjugation and localization between pi groups and the presence and types of acceptor and donor groups.

While our own feature selection gave us a preliminary measure for the type of structure we were dealing with, we needed a means to more accurately identify the substructure features discussed above. To obtain these features we employed the Morgan Fingerprint, a circular substructure search aglortihm which generates unique identifiers for molecules starting at radius 0 (the molecule itself) iteratively up to a specified radius taking into consideration certain invariants of atom within each iterative radius. It then compresses these hashed identifiers into a bit vector of specified length.

The parameters we chose for our initial Morgan fingerprints were a radius of 2 and a fingerprint size of 512 bits. We felt most of the interesting structural features could be captured within this radius, while 512 was an optimal bit size as it captured key features while reducing dimensionality. The default atomic invariants were enough to capture much of the HOMO-LUMO features.

As a secondary test we employed the Morgan algorithm to generate another 512 bit feature list using the Pharmacophore features of the atoms and conjugated it to our existing feature list. We though it might produce additional structural detail not captured in the atom invariant algorithm. This produced a marginal increase in predictive power (RMSE = 0.5044) but increased the computational time exponentially. If we had more computational power and memory we would have evaluated: the effect of extending the Morgan

Fingerprint Radius and Fingerprint Size *and* the performance of linear substructure search algorithms against the circular Morgan fingerprinter.

1.2 Model Classes

Ridge Regularization

As we discussed in class, there is a *bias-variance tradeoff* dependent on the complexity of a particular model; as a model becomes more complex, it tends to fit the training data better (thus reducing bias) but then may experience overfitting and large swings in parameter values depending on the specific data it is trained on (increasing variance).

As such, given a large feature set (or high-dimensionality), linear regression may encounter problems related to overfitting. To combat this, we used regularization, a mathematical adjustment to the model loss function intended to penalize model complexity. In the case of Ridge regression (also known as L_2 regularization), the modified loss function becomes:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + \frac{\lambda}{2}\mathbf{w}^\top \mathbf{w}$$

where λ is the regularization hyperparameter, influencing how much the loss function penalizes extreme parameter values. As such, smaller λ values make Ridge regression imitate normal linear regression while larger values will lead to significantly smaller parameter values. To choose a value for λ, we used a built-in *sklearn* function called `RidgeCV`, which conducts k-fold cross-validation (we chose $k = 5$ as is convention) to choose the best value for the hyperparameter given out-of-sample performance. `RidgeCV` actually tunes $\alpha = \frac{1}{\lambda}$ which we kept track of during our analysis.

Random Forest

Random Forest is an ensemble method based on a Decision Tree. A Decision Tree splits up the feature space (in d dimensions) into locally linear boundaries. It uses a greedy algorithm with the following steps: start with an empty decision tree, choose an optimal predictor on which to split and an optimal threshold value, recurse on each node until a stopping condition is met.

For regression, the splitting criterion is mean squared error; as such, the new split chosen is the one that will minimize total variance across all regions (weighted properly). Mathematically, a decision tree minimizes:

$$Err(T) = \frac{1}{n}\sum_{l \in \tilde{T}}\sum_{D_l}(y_i - k_l)^2$$

where l is a leaf in $\tilde{T}$, the set of all leaves in tree T. As you can see, it's essentially minimizing the total weighted variance across all leaves.

Decision trees can usually only reduce the bias sufficiently when the tree is deep enough to lead to high variance/overfitting, which is why we use a Random FOrest. We bootstrap many decision trees (depending on the `n_estimators` hyperparameter) and average the output from each one for predictions. In order to de-correlate trees, and thus reduce the likelihood of overfitting, at each split we choose random groups of predictors (how many are available is based on the `max_features` hyperparameter).

Thus, we can reduce both the bias and variance associated with single decision trees using the Random Forest method, and *threshold values become even more valuable/intuitive given the binary features in our data set.*

AdaBoost

AdaBoost is a type of gradient boosting algorithm, creating a strong model from a number of weak regressors. Unlike random forest, where the individual trees are generally uncorrelated from each other, each new tree in an AdaBoost ensemble is fit to the residuals of the previous one.

As such, it iteratively builds a more complex model via simple models in the following way:

1. Fit a simple model, $T^{(0)}$, and set $T \leftarrow T^{(0)}$. Then compute the residuals.
2. Fit a simple model, T^{i}, to the current residuals. Set $T \leftarrow T + \lambda T^{i}$.
3. Compute residuals, setting $r_n \leftarrow r_n - \lambda T^{i}(x_n)$ for $n = 1, \ldots, N$.
4. Repeat steps 2 and 3 until stopping condition met.

Here λ is the learning rate, an important hyperparameter called `learning_rate` in *sklearn*; this can be solved using gradient descent, but in our case we tuned the hyper-parameter using cross-validation. Higher λ tends towards more overfitting. We also looked at the number of trees and the maximum depth of each tree as important hyper-parameters. Because of the iterative nature of AdaBoost, it can be more prone to overfitting than Random Forest and this influenced our view of the algorithm as well as its slow run-time.

2 Results

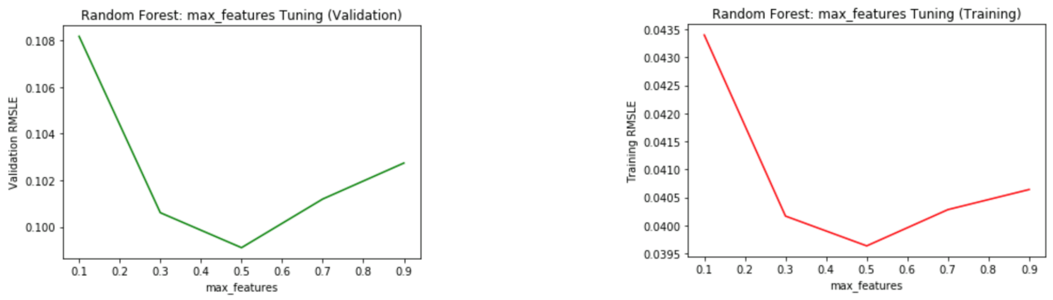

Figure 1: Effect of `max_features` hyperparameter on Random Forest RMSE results.

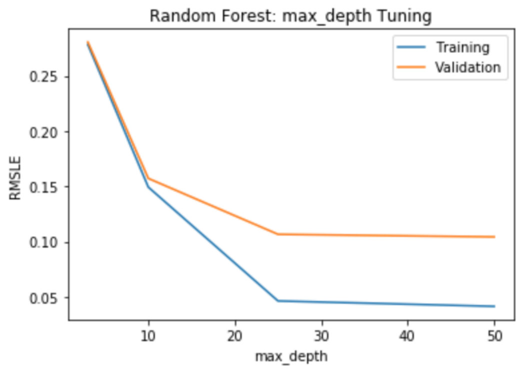

Figure 2: Effect of `max_depth` hyperparameter on Random Forest RMSE results.

Model Number	Model Description	Tuning?	Test RMSE Result
0	Random Forest with baseline features	No	.27209
1	Random Forest with Basic RDKit	No	0.16
2	Ridge with Morgan	Yes	0.155
3	Random Forest with Morgan	No	0.1
4	Adaboost with Morgan	No	0.1345
5	Random Forest with Morgan	Yes	0.05282
6	Random Forest with Morgan and Pharmacaphore	Yes	0.05044

Above are our model results on Camelot's partial test set. As you can see, our best model far outperformed the baseline in terms of RMSE.

3 Discussion

We iteratively chose models that were more complex in one of two ways: either in the number of features or the class of model itself. In Model 1, we performed a Random Forest with the basic features. We then added the Morgan features and used this dataset in Models 2-5.

The Ridge Regression (Model 2), which was run on $\lambda = \frac{1}{2}$ based on cross-validation results, was the simplest and, as we expected, performed the worst. The relatively small λ made sense, as in this case the number of molecules far outnumbers the amount of features we have. In Model 3, we tried the Random Forest Model, without tuning our hyperparameters (`max_depth`, `n_estimators`, and `max_features`). Model 4 used Adaboost without tuning our hyperparameters.

Model 3 ended up being the best, so we focused further work on the Random Forest. The fact that the Random Forest worked best fit with our intuition about the ideal nature of binary data for a process which splits up the feature space into locally linear boundaries. Ridge tends to work relatively poorly with binary features, while Adaboost was computationally slow, tended to slightly overfit, and struggled with the rows (data points) not being particularly similar (thanks to TF Ragu for helping us understand that).

We then tuned RF's hyperparameters using `GridSearchCV`. See Figures 1 and 2 for results from tuning. As you can see, we found that a middle value for `max_features` and a larger `max_depth` performed better out of sample, which drove our intuition on a final RF model to use. We used these graphs and further cross-validation in order to help us decide on our eventual hyperparameters, which ended up being `max_depth` $= \infty$, `n_estimators` $= 250$, and `max_features` $= 0.6$. This tuning resulted in near halving of the RMSE (from .1 to .053). Finally, we added the Pharmacaphore features to our data, and the resulting model produced slightly better results using a Random Forest.

In general, the Random Forest model outperformed both the Ridge and Adaboost models. In addition, adding more features helped our results at each step of the way, as did tuning the hyperparameters. These are all intuitive results because the random forest does not suffer from overfitting, so it was generally helpful to add more features, and because the random forest works well on binary feature sets.