CS 181 – PRACTICAL 1

NAI CHUN CHEN , SHWETA MEHTA, WENFEI XU

1. TECHNICAL

Unlike silicon-based solar cells, their carbon counterparts are cheap to produce, flexible, and transparent. However, all known carbon-based solar cells have a lower efficiency than the silicon-based ones. The motivation for our work is to predict and hence identify the most efficient carbon-based solar cells using machine learning techniques which is computationally faster than other methods in the market.

1.1. **Data.** We were given a training dataset of a million organic molecules and a test dataset of another 800,000 molecules with their molecular structure represented as a Simplified Molecular-Input Line-Entry System (SMILES) string. In addition, we were provided with a set of 256 binary features characterizing these molecules. The primary quantity of interest is the 'gap' energy which is the difference in the energy between the highest occupied molecular orbital (HOMO) and the lowest occupied molecular orbital (LOMO). The goal of this machine learning exercise is to predict the 'gap' energy of each molecule using their corresponding features; the lower the gap energy, the higher the potential solar panel efficiency.

1.2. **Features.** The first thing we observed was that the 256 features provided to us as part of the dataset were in a binary format and hence characterized with a low variance. This is not ideal for prediction purposes and a basic linear as well as a random forest regression model justified our thoughts. We realized that feature engineering would be a big part of the challenge and hence we carried out a two-approach strategy for feature engineering.

The first approach was to use the binary features to generate additional features. Out of the 256 features given to us, we identified 30 features which had some non-zero values, and only used these to create additional features. New features were generated by multiplying and adding two columns with one another. Hence we generated $2 * \frac{C_2^{30}}{2} = 870$ new features. Our intuition was to counteract the binary and low variance of the original feature set by adding more features from RDKit with wider ranges so our model would have more information per molecule to assist in prediction.

Our second approach was to use the SMILES code and the molecular structure to generate 124 additional features. 114 of these were generated using a chemistry package called RDKIT in Python which reads SMILES strings to produce features describing the molecular structure. An example would be the number of aromatic rings, molecular weight, and surface area. The remaining 10 features were created via a string manipulation of the SMILES code. Based on a manual observation we noticed that the molecules with the lowest 'gap' energy had a certain, reoccurring pattern of sub-atomic groups such as 'nH', 'c1sc', and 'SiH2', among others. We generated new features by counting the number of such patterns in each molecule. We also counted the number and type of bonds in each molecule. This helped us generate a total of 124 non-binary features which had a large variance to assist in our prediction modeling.

Finally, after we plotted the distribution of the gaps, we could see that there was some periodicity in our data.

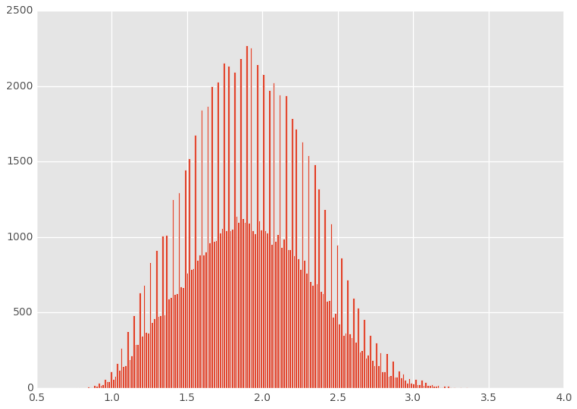

FIGURE 1. Distribution of HOMO-LUMO gap.

From this observation, we attempted a sinusoidal basis transformation in which our model class was:

$$w_0 + \sum_{n=1}^{N} w_n X_n + \sum_{j=1}^{J} \sum_{n=1}^{N} w_n sin(j * X_n)$$

Because we took into account the periodicity, the RMSE was more accurate in the case of a sinusoidal basis transformation. However, this also created J times more features, which increased the run-time of our algorithms. Ultimately, we decided against running this transformation as the improvement was not great.

1.3. **Initial Methodology.** As a first pass we used the original 256 features and the 870 binary features created as part of 'approach 1'. We ran Principal Component Analysis (PCA) which is a-priori feature selection to help us reduce the dimensionality of the problem. It helped reduce our large feature set to 50. We ran linear regression on this dataset without observing a significant improvement in our overall prediction capabilities. Hence we discarded these features.

The next step was to use the more descriptive feature set containing the original 256 features as well as the additional 124 features created using RDKit and string manipulation.

We divided our training data into 'train' and 'validation' sets. The 'train' set would help us fit the data and tune our parameters. We performed ten-fold cross-validation on the 'train' set to select the best parameter which produced the lowest error. Finally the 'validation' set was used to evaluate the performance of our models on fresh (aka 'never-seen-before') data. We did this by fitting our classifiers on the 'validation' data and calculating the root mean squared error (RMSE) as a way to compare and pick the best model.

We first applied Linear Regression on our 'train' dataset. Linear regression is a method to model the relationship between a set of independent variables X (i.e., explanatory variables or features) and a dependent variable Y (i.e., target). This method assumes the relationship between each predictor X is linearly related to the dependent variable Y. It estimates the coefficients of the linear

regression equation to minimize the error between the predicted and actual values.

One concern while using unregularized Linear Regression is overfitting the data. In such a scenario, we explored regularization as an important technique to prevent overfitting since it would allow us to to smooth or regularize our function by reducing the complexity of the model. Ridge regression, a closed-form method of penalizing the 2-norm of the regression weights, was first used. Since the predictions were not too different from unregularized Linear Regression, we used Lasso Regression. Lasso Regression is another form of Linear Regression, which penalizes the 1-norm of the weights. In Lasso Regression, the regularization penalty is proportional to the sum of the weights, which tends to induce sparsity and pull the weights of the least useful features to zero before affecting the weights of the other features. While carrying out regularization, we also performed parameter tuning to choose the best penalizing factor 'alpha' which results in the least error. For ridge regression we were trying to find an alpha to minimize the mean squared error, whereas for lasso regression we were trying to find an alpha to minimize the absolute mean error.

Since the performance of regression methods was not satisfactory, we tried an ensemble method, Random Forests. Ensemble methods are a set of techniques to combine several weak classifiers together to build very accurate predictors.

We used the scikit-learn library in Python for cross-validation, parameter tuning and to carry out Linear, Ridge, Lasso, and Random Forest Regression.

2. Results

2.1. **Linear Regression.** After the addition of our new features, our $RMSE$ improved quite a bit. The result from simple linear regression improved from the original 0.27 to $RMSE = 0.1899$. Looking at the plot of predicted vs our validation set, we could find the values under 1.0 and above 2.5 are not being accurately predicted.

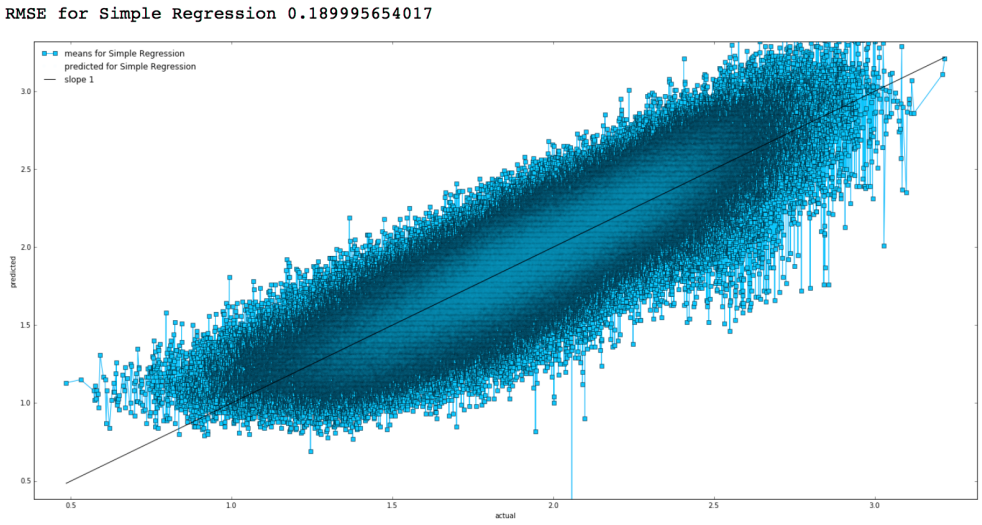

Figure 2. Predicted vs Validation set for Linear Regression with additional features.

2.2. **Ridge Regression.** In ridge regression, we tried different $\alpha \in 0, 1, \ldots, 100$, which then allowed us to narrow down an $\alpha \in 0, 0.0001, \ldots, 1$. Then we picked the best one by cross validation with 10-fold.
We got that $\alpha^* = 0.0001$ and $RMSE = 0.18999$. The Ridge regression does not improve a lot in prediction. We still have issues in prediction is lower than actual values when gaps are under 1.0, and prediction is higher than real values when gaps are above 2.5.

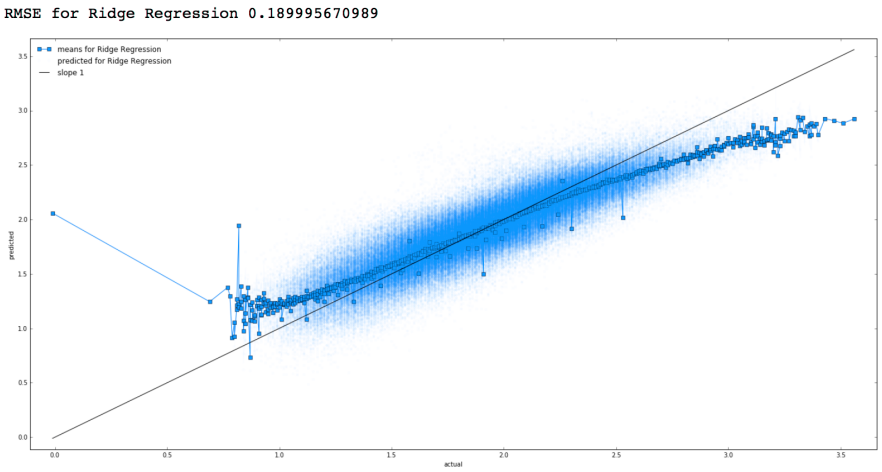

Figure 3. Ratio of Ridge Regression and Actual Value.

2.3. **Random Forest.** With the random forest regression process, we used a cross-validation with a 70-30 breakdown between our training and validation set. We used $N_trees = 10$ with the standard decision tree regressor criteria in SKLearn. The Simple Random Forest $RMSE = 0.1462$. The Root Mean Squared Error decreased 0.04. For better prediction, we also tried forests with different numbers of trees.

(1) N = 1 : 0.151
(2) N = 6 : 0.1039
(3) N = 11: 0.0984
(4) N = 16: 0.09718

The more tree we use, the better prediction we have. In the final submission, we use tree=16, and got 0.097. The figure of Ratio of Random Forest Prediction and Actual Value also shows in 45 degree line, means the prediction is pretty good.

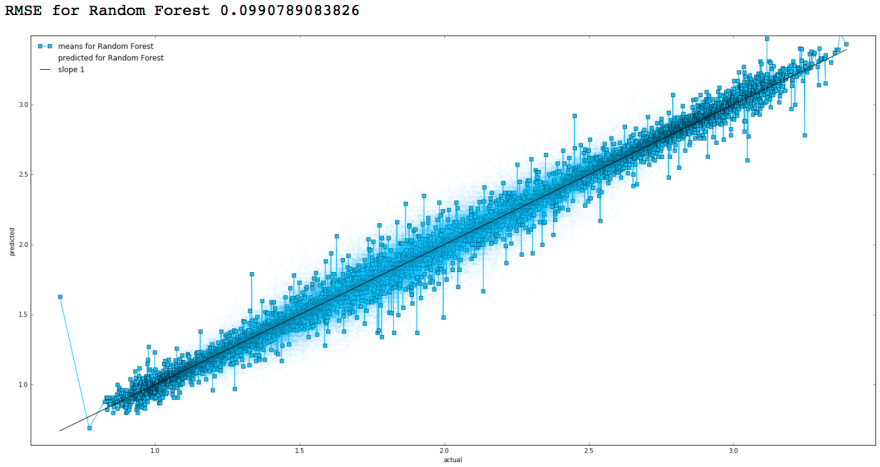

FIGURE 4. Ratio of Random Forest Prediction and Actual Value.

Both of the Error Distributions of Random Forest and of Regression are normal distribution, but the Error Distributions of Random Forest is more centralized than Regression's Distribution.

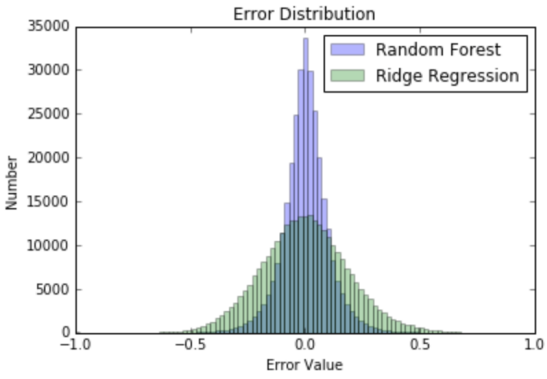

FIGURE 5. Error Distribution between Random Forest and Ridge Regression.

3. Discussion

3.1. **Feature addition and preprocessing.** Our process was geared towards faster initial calculations that would allow us to calibrate our models better and pick the best method to fine tune. Thus, as discussed in earlier sections, we first started with a simple linear regression and a 70-30 breakdown of train-validation datasets from our training set. From this initial result, we got $RMSE = 0.27$. We were left with 379 features after we added features from RDKit.

When we added additional non-binary features, we were able to increase the variance of our data and, as can be shown from the feature plots below, we were also able to give some directionality as well.

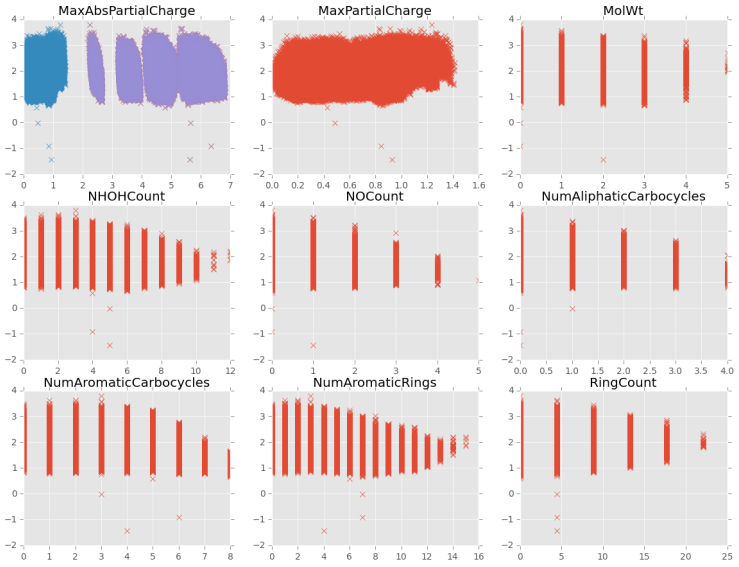

Figure 6. Selected features from RDKit plotted against the HOMO-LUMO gap.

3.2. **Regressions.** When we ran regressions on the original dataset, we saw that our coefficients were on the order $\pm e^1 0$, which led us to believe that we could have an over-fitting issue (and hence the attempts at ridge regression. However, when we included the additional features from RDKit, the magitude of the coeffcients was reduced due to the fact that our features were providing more explained variance. With the new dataset, we ran a ridge regression when we saw that our coefficients were mostly on the order of $e^{\pm 1}$ and $e^{\pm 2}$. And when we tested for an optimal α, the resulting α^* was actually quite small.

We also ran a lasso regression to reduce the number of important weights we had, as our feature class was fairly large. However, this did not greatly improve the $RMSE$.

3.3. **PCA transformation.** Because we had a large feature set, we then performed a PCA transformation on our set in order to reduce the dimensionality of our feature matrix, which improved the performance speed of iterative processes such as decision trees and ensemble learners later on.

3.4. **Tree and Ensemble Learners.** The reason why tree performance is much better than regression is that a lot of columns in the training data sets are only binary data (0,1). Tree structure methods , including decision tree and random forest, are also binary, could have better performance in binary data. We could use regression to find columns with higher correlation and use the features selected by regression to prediction "gaps" with Random Forest.

4. REFERENCES

.
1. Python scikit-learn library
2. Harvard CS109. 2015. Data Science. Lab Notes.
3. RDKit. Open-Source Cheminformatics Software. Available at: www.rdkit.org
4. Tingley, Michael Alan. 2014. Towards the Quantum Machine: Using Scalable Machine Learning Methods to Predict Photovoltaic Efficacy of Organic Molecules. Bachelor's thesis, Harvard College.