# Practical 1 - WORKING DRAFT

Samuel Daulton, Taylor Killian, and Andrew Petschek (ML Marauders)

February 10, 2016

## 1 Technical Approach

### 1.1 Initial Attempts

Our initial approach to expand on the sample code provided to us for this machine learning problem was to implement Principal Component Analysis (PCA) to gain a better understanding of the interdependencies between features and to effectively reduce our feature set. We also performed some basic feature engineering upfront to couple with the PCA. There is a hypothesis that the number of branches influences the gap between HOMO and LUMO levels, and we pulled relevant compounds that were highlighted in a theoretical study on this topic conducted by Padtaya Pukpayat of the University of Nevada Las Vegas. This specific feature aggregates the number of active features, meaning if there are more features that are active in a compound, then there may be a higher orbital overlap and, thus, a lower HOMO-LUMO gap. A final initial feature we added was the presence of a benzene ring in the compound; we could identify a benzene ring which appeared to be more conjugated (i.e., closer together in energy). We combined this initial feature generation with PCA, and turned to the existing methods to see if it we gained improvement over baseline. The results were promising, but we decided additional, more robust features needed to be engineered for full impact.

### 1.2 Feature Engineering

To create new features, we turned to the RDKit. Specifically, we leveraged molecule object methods that returned integers. We felt integer outputs would work best with our modeling efforts, and we avoided methods that would output that non-integers (e.g., strings and vectors). 114 new features were created in this way that spanned four main areas: general feature generation (e.g., number of branches, number of bonds), descriptors (e.g., molecular weight, number of valence electrons), calculation of Lipinksi parameters,

and fragment descriptors (e.g., number of aliphatic carboxylic acids). As stated above, these methods act on molecule objects, which take up a lot of memory, so our technical approach to feature generation was to divide the processing into four separate parts, which were ultimately merged together at the end.

In addition to using RDKit, we implemented our own algorithms to additional new features. We had a hypothesis that there likely existed interactions between features that would lead to larger HOMO-LUMO gaps. Thus, in addition to including a linear basis for each feature, we aimed to also include the product of pairs of features as well. Since we had well over 100 features, we decided we would look at the combinations between the top 25 most important features. To determine the top 25, we turned to Random Forests, which can output the importance weighting of each feature in our feature space. Using this regression output, we took the top 25 and created $25C2$ (or 300) new features, which were the products of two features within an unique combinatoric pair.

A final method for feature generation we used was based on logistic regression. Performing first some pre-processing, we rounded each gap value to the nearest half integer which we set as the labels we wished to predict. We ultimately used these labels as categorical features that were included in our overall feature set.

### 1.3 Modeling Techniques

We not only built off of the simple linear regression and random forest model classes provided to us, but also approached this machine learning problem from other modeling angles as well. Across all methods, to prevent overfitting out-of-sample data sets when evaluating various hyperparameters for estimators, we split our training set into a training set and a validation set (67-33 split). This way, we avoided any knowledge of the test set leaking into our models and hurting their ability to generalize to new data. We had initially contemplated a cross-validation loop, where the creation of discrete training and validation sets would no longer be necessary to increase training sample size. However, since we had roughly one million samples to begin with, we decided we would have sufficient data to carry out our modeling using discrete training and validation sets. When estimating hyperparamters for each of our models below, we employed a grid search. Since we did not know which hyperparameter values may minimize our loss functions across models, we ran each model several times across an array of hyperparameter values ranging from small to large. At the end of the loop, we selected the hyperparameter value(s) that yielded a minimum loss when tested against our validation set.

Examples of this grid search method were evident in three new methods we employed: Lasso Regression, Ridge Regression, and Elastic Net Regression. Lasso regression is a linear model with a regularization term (i.e., L1 prior). The constant, $\alpha$ that multiplies

the L1 term is a hyperparameter that can range between 0 and 1. We noticed that if $\alpha = 0$, then we have reduced this regression back to a simple linear model without regularization. Ridge Regression is similar to Lasso, except it uses a L2 prior. Similarly, if $\alpha = 0$ in our Ridge Regression, we have reverted back to linear least squares. Elastic Net can be thought of as a combination of Lasso and Ridge Regression in that it combines both L1 and L2 priors as regularization terms. There are two hyperparameters in Elastic Net which measure the relationship between the L1 and L2 penalties. Here, if $\alpha = 0$, we again have a linear least squares regression and if the ratio equals = 0 we have only Ridge Regression and if the ratio = 1 we have only Lasso Regression. Thus, we employed grid search to determine the right balance needed for loss minimization. With similar results across all three, we ultimately landed on using Lasso Regression for our final submission. Lasso regression is able to zero out features completely while Ridge Regression, for example, cannot. For this reason, we decided Lasso Regression may be a better bet to generalize to the new test data set moving forward since we had hundreds of features.

In addition to linear modeling, we also employed ensemble methods, including Random Forests and Extra Trees Regressors, which can be thought of as a close alternative to a Random Forest. In our Random Forest regression we set the number of features per tree to be equal to $\frac{1}{\sqrt{p}}$ and used a grid search to compare number of estimators between 10 and 128. We had read in THIS PAPER that a good range of estimators to be 64-128, as anything more would result in additional computation time with minimal gains. which is what dictated our rationale for the grid search bounds. We repeated this similar approach for Extra Trees Regressors. As stated above, Extra Trees is similar to Random Forests, where Extra Trees randomly draws estimators from the entire training set instead of a bootstrap sample and these splits are chosen at random. This additional set of randomness can lead to a reduction in variance and an increase in bias. For this reason, while we did run tests using Extra Trees Regressors, we ultimately decided to use a Random Forests instead for their potentially lower bias and better ability to generalize to new data.

A few extensions were used across the methods above. The first involves PCA. We had initially thought that the use of PCA would be beneficial, as it would reveal the most important features in our feature space up front. We thought that this would improve performance. We found, however, that this did not seem to be the case. Reducing our feature space using PCA did not seem to improve our predictions (and in some cases it even led to negative impact). We also considered standardizing all non-binary features to ensure they are normally distributed. We felt normally distributed data would potentially lead to more accurate predictions. This too, like PCA, seemed to lead to worse results. Thus, we ultimately decided not to standardize our data.

DO WE WANT TO ADD ADABOOST, GRADIENT DESCENT? Finally, we also built upon this ensemble method by incorporating AdaBoost. AdaBoost fits a sequence of weak learners on repeatedly modified versions of the training set. The predictions are assigned

| Model | RMSE (Public Score) |
|---|---|
| Extra Trees Regression and PCA (n=60) | 0.15337 |
| Lasso and PCA (n=60) | ??? |
| Extra Trees Regression and PCA (n=45) | 0.22972 |
| Extra Trees Regression with AdaBoost | 0.29581 |
| Linear Regression | 0.3 |

Table 1: Model Results

weights which update through a loop. These weights were measured according to two loss functions: linear and squared error. Together, Adaboost with Extra Trees were used.

## 2 Results

## 3 Discussion

This was an exciting opportunity to be able to work with a complex dataset on a machine learning problem with real-world implications. We approached this task as a prediction problem rather than an estimation problem. While prediction and estimation are similar, there is nuance. While estimation concerns itself more with the structural properties of the model and a more physical understanding and interpretation of coefficients, prediction cares most about loss minimization and the models ability to generalize to new data. Thus, we approached this problem with an open mind and a determination to find a minimal loss score without preference for the type of model we employed.

While we did not find PCA to help much with our predictions, it did remind us of a hallmark of scientific computing. PCA reduced our feature set and greatly reduced computation time. While computation time is not a benchmark we are graded on, it is an integral component to scientific computing. As data sets become large, computation time required increases. Thus, one's ability to effectively reduce the time it takes to fit a model without sacrificing accuracy is highly valued. While PCA was not effective here, we could have considered additional parallel processing techniques (e.g., multi processing, multi threading) to speed up computation time.

We found feature engineering to be a critical component to our success. Our modeling output is only as good as the input we provide. We found that our feature engineering efforts provided our model with regression input it needed for better training. With each iteration of feature generation, we found our results to increase.

Further, we believe an ensemble method of models to be the most robust with such a complicated data set.

add more