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 calculated the top thirty to sixty components, and compared which number of principal components appeared to be best from the start.

We also performed some basic feature engineering upfront as well 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 used was the presence of a benzene ring in the compound using the SMILES encoding; we could identify a benzene ring which are 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 improved baseline. The results were promising, but we decided additional features needed to be engineered for full impact.

1.2 Feature Engineering

To create new features, we turned to the RDKit. Specifically, we leveraged methods that process molecules objects and return integers. We felt integer outputs would work best with our modeling efforts and avoided outputs that non-integerse (e.g., strings and vectors).

114 new features were created 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). The methods that generated these features act upon molecule objects, which we created using the RDKit API. Molecule objects take up a lot of memory, so our technical approach to feature generation was to divide the processing into four parts, which were ultimately merged together at the end.

In addition to using RDKit, we implemented our own algorithms to generate new features. The first involved the use of random forests. We set the number of features per tree to be equal to $\frac{1}{\sqrt{p}}$ and set the number of estimators to be 128. We had read in THIS PAPER that a good range of estimators is 64-128. Anything more would result in additional computation time with minimal gains. We used this technical methodology in our modeling techniques as well (see below). Using this regression output, we were able to deduce which features available were deemed to be most important. We took the top 25 and created 25C2 additional columns. We hypothesized that there are likely interactions between features, thus we created new features that were the product of 25C2 features.

A final method for feature generation was that of logistic regression. We performed logistic regression on the gap values, however we performed pre processing and rounded each sample to the nearest half integer. This way, we could provide a label to our data along the lines of "this is what a 1.5 gap sample looks like". We then aimed to use this classification labels as features for our modeling.

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.

To prevent overfitting on 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. 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.

When estimating hyperparamters for each of our models below, we employed a grid search. Since we do not know which values of hyperparameters may minimize our loss function across models, we run each model several times using an array of hyperparameter values

ranging from small to large. At the end of this loop, we selected the hyperparameter value(s) that yielded a minimum loss.

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 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. Since we have many features, Lasso regression is able to zero our some features 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.

In addition to linear modeling, we also employed ensemble methods, including Random Forests and Extra Trees Regressor. Extra Trees Regressor can be thought of as a close alternative to a Random Forest. In our Random Forest regression we attempted three variations. The first was leveraging PCA to determine which features were most important and reducing our feature set. We found this to be disadvantageous as random forests perform better with more features. PCA did not seem to help much with our ability to predict. Another variant on our Random Forest regression was applying a standardization to our non-binary features. We aimed to normalize all inputs for hopefully better results. However, this too did not seem to aid much. Thus, we settled on no PCA and not standardizing our data prior to fitting our model. 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 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.

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 leads to reduction in variance but may lead to increase in bias. For this reason, we ultimately decided to use a Random Forests instead of Extra Trees for their potentially lower bias and ability to better generalize to new data. We attempted similar permutations of the above (e.g., PCA and standardization)

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

for trees equal to 10, 32, 64, and 128.

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