Practical 1

Samuel Daulton, Taylor Killian, 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. As stated in our results section below, PCA led to satisfactory reduction of RMSE.

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.

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.

Model	**RMSE**
Random Forest	0.2
Linear Regression	0.3

Table 1: Model Results

In addition to linear modeling, we also employed ensemble methods, including an Extra Trees Regressor. Extra Trees Regressor can be thought of as a close alternative to a Random Forest. Extra Trees randomly draws estimators from the entire training set instead of a bootstrap sample and these splits are chosen at random. We carried out Extra Trees Regression with and without our insight from the PCA. We then 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.

Finally, we

1.4 Lasso

1.5 ExtraTrees

1.6 Adaboost with ExtraTrees

1.7 Steepest Descent

2 Results

3 Discussion