CS 181 Practical 1: Predicting Chemical Toxicity for Drug Design

Meriton Ibrahimi
meritonibrahimicollege.harvard.edu
Kaggle: @meriton

February 16, 2019

1 Technical Approach and Discussion

We begin our approach by first looking at the data. The data contain greater than 251 drug features. All the features appear to be in scale from 0 to 1. Importantly, a number of the features, upon closer examination, are constant i.e. each drug has the same value for theses features. Because these features offer no predictive power and because such features may even hinder performance on training algorithms, we choose to remove them. With those uninformative features removed, we proceed to randomly split our training data into data we will train our algorithms on and a validation set to provide a measure of performance. Examining our data, we plot the features against our target column. Examining these plots, it is difficult to immediately determine a relationship between the predictor features and our target. We see that the target values roughly cluster around values form 0.85 to 1, and if we plot the target as a series, we see a flat plot with seemingly two bold lines running at values of 1 and at values between 0.9 and 0.925.

1.1 Linear Regression

We begin our approach with good-ol' linear regression. This classic model asserts that our target - within some noise - can be determined as a linear combination of our predictor features. We fit this model and notice very large coefficients for our predictors. This is curious as our target is on the same scale as our predictors. This hints at the potential instability of the linear regression model. In our prepossessing of the data, we were able to remove potential extreme sources of instability in the constant, uninformative predictors, but the other features are also heavily correlated. Our suspicion for a poor fitting model is confirmed as we test model our held-out validation set, reporting a very excessive root-mean squared error validation score. To address these concerns in our predictors and coefficients, we move onto regularization, where we set limits on the magnitude of the coefficients.

1.1.1 Ridge

Ridge regression corresponds to adding a L2 norm to the original optimization function in linear regression. Ridge Regression has the effect of shrinking predictor coefficient towards zero. This is

why this model is also called a shrinkage method. Importantly, Ridge Regression does not shrink coefficients completely towards 0 (which is a product of the optimization method). In fitting this type of model, we need to determine by how much we should regularize. The Ridge Regression optimization function is a linear combination of two sub-functions and the weight on the L2 loss determines the amount of regularization - we call it lambda. To determine the optimal value of lambda, we perform 20-fold cross-validation over several values of lambda ranging from 0.01 to 100. Through this we obtain a chosen lambda of 25. As a crude measure of shrinkage, we see that the sum of the absolute values of the Ridge coefficients are about 0.5, whereas this value in the previous, standard linear regression was on the order of hundreds of billions. In testing this Ridge model with our chosen lambda on our validation set, we achieve a RMSE of 0.027125.

1.1.2 Lasso

Like Ridge Regression, there is a sister form of regularization which adds an L! norm to the original linear regression optimization. Unlike the optimization in Ridge Regression, the form of the Lasso optimization can shrink coefficients all the way towards zero. Like Ridge Regression, we need to determine the optimal amount of regularization. We do this with 20-fold cross-validation over a number of lambda values, attaining a chosen lambda of 100. Very curiously, this "optimal" lasso model allocates all predictor coefficient to 0. The model instead favors a fit with just a constant term at 0.91951. From the plot we saw in examining our data, this makes sense. When the target is plotted as a series, we see the data cluster around a flat line between 0.9 and 0.925. In testing this model with our chosen lambda on our validation set, we achieve a RMSE of 0.0.027994.

1.1.3 Elastic Net

Elastic Net is a form of regularization which combines Ridge and Lasso together, adding both a L1 norm and L2 norm to the original linear regression optimization. Here, we have two parameters to validate over: the amount of each type of regularization to add and the ration between these values. In performing cross-validation, we achieve a model equivalent to the Lasso.

1.2 GLM

In regressing our predictors onto our target, we notice that we are not necessarily guaranteed predictions that will fall in line with the range of our target. The linear combinations of predictors may produce values outside the range 0 to 1 unless we specifically set a model to avoid this. Here, we apply a generalized linear model. Linear regression is a specific form of GLM where the link function between the linear combination of predictors and the target is the identity link. Because our target is on the range from 0 to 1, we can use the logit link function. What this essentially means is fitting a linear regression model with the target column transformed by the logit function. We make the transformation and carry out linear, ridge, lasso, and elastic net regressions. Unfortunately, none of these do any better on our validation set, most likely because when we make the transformation we introduce very influential variables - target values of 1 cannot be transformed by the logit functions so we intentionally add some noise but this introduces outliers in the transformed space.

1.3 KNN

Parameterized models seem to have reached their limit in a dataset with no clearly apparent structure outside a flat line, so we turn to a non-parametric model, K-Nearest Neighbors Regression. What this model does is for each test data point, the model will find the k "closest" points in the training data to that test point and return the average of these training data as the prediction. In this model, what needs to be optimized is the value of k. We run the KNN model over values of k ranging from 1 to 750. We find an optimal value of k around 10. At this value, we reach peak performance of the model without running too large a risk of potentially overfitting. We achieve a validation RMSE score of 0.02729.

1.4 Trees

We move our focus onto another non-parametric model, Decision Trees. Decision trees work by continually finding splits in the dataset on specific values of a predictor (e.g. if predictor x is less than value v) such that you build a tree in the sense that if we follow the simple if-then statements of these splits down the tree, we reach subsets of the data that we can average over to output a prediction. These splits are made such that it produces two regions of the data with the smallest combined scoring criterion - usually mean-squared error. In this model, one key hyperparameter which needs to be optimized is the depth of the tree. To do this, we perform 20-fold cross validation. In doing this, we obtain an optimum depth of 2. Again, this is curious but intuitive. As we explained previously with lasso and the initial data exploration portion, the data seems to be clustered around two flat lines at a target value of 1 and at a target value between 0.9 and 0.925. Indeed, this is how the decision tree is making its splits. Using this model, we achieve a validation score of 0.0275.

1.4.1 Bagg of Trees

We see from above that the Decision Tree reaches an "optimal" amount for our cross-validation score at a given depth. By doing this, we attempt to reach a minimal loss in the bias-variance trade-off. In bootstrap aggregation of Decision Trees, we fit many overfit decision trees on bootstrapped samples of our data and make predictions based on the average value output by each of the tress in our bag. Each tree thus has low bias and high variance, but we expect the average of the trees to have low bias and low variance. We can visualize the effect of the number of trees on the RMSE of the entire model, but we see that around over 50 Decision Trees, we reach a plateau. For each overfit tree, we use a depth of 3 times that found in the cross-validation approach above to ensure we are overfititng. In running this model, we achieve a validation RMSE score of 0.026815.

1.5 Random Forest

Random Forests are an extension of bootstrap aggregation of decision trees. In bagg of trees, although we boostrap the original dataset to build a new decision tree on each iteration, a problem can occur in that we may have equivalent trees being added to the bag if a single predictor is always chosen as the best predictor on which to start splitting (this can also happen further down

Model	RMSE on Validation Set
Baseline Linear Regression	1631359608.852832
Ridge Regression	0.027125
Lasso Regression	0.027994
Elastic Net	0.027994
Ridge GLM	0.029929
Lasso GLM	0.030955
Elastic Net GLM	0.030814
KNN	0.027289
Decision Tree	0.027512
Bags of Tree	0.026815
Random Forest	0.026733
AdaBoost	0.027861
Neural Net	0.027464

Table 1

the tree as well). Thus, you run the risk of producing similar trees that will not aid in reducing the variance of your overall model. What the random forest algorithm does is, at each split of the growing tree, randomly choose a sub-sample of the predictors from which to split on. This has the affect of producing different trees each time. Thus, when running a random forest algorithm, we need to optimize over two hyperparamter: the number of trees and the number of predictors over which to randomly select at each split. Cross-validation over these hyperparameters, we achieve a RMSE score of 0.026733 on our validation set.

2 Results

Results of all our models are displayed in Table 1 along with results of a boosting model (AdaBoost) and Neural Net (explained in the IPython notebook). On our validation set, the random forest outperformed all. We submitted several entries to kaggle predicting on the test set. The model which performed the best on kaggle was the neural net with three hidden layers of 208 nodes (the number of features excluding the uninformative ones). We attempted to optimize this Neural Net many times by playing with the architecture. We ran deep layers of 208 nodes with the number of hidden layers approaching 10. We aimed to try architectures with increased number of nodes in the middle relative to the beginning and end of the neural net; we went the other route as welll, decreasing the number of nodes in the middle. We must have run several tens of combinations with varying architectures, optimizers, etc. and came to the best architecture as noted. It seems that at some point, we reached a limit as to what we could extract from the data. If we were to continue examining this problem, I would consider reducing the dimentionality of the data as some of the features are very correlated. Reducing the dimentionality might also have the affect of making certain trends and patterns in the data more apparent.