Introduction and Results

I did an experiment to see how feature neutralization affects the bias and variance of a model. This involved fitting many models on eras that were randomly sampled from the training set, and using the validation set to calculate bootstrap estimates of bias and variance from the resulting predictions. This was repeated with several neutralization proportions to see which values were ideal for trading off bias and variance. See the Kaggle notebook for implementation details.

The results show that as the neutralization proportion increases, bias increases and variance decreases. The sweet spot for balancing bias and variance seems to be around $p = 0.5$

for the model that I used.

Although it’s quite slow, this method could be an alternative or supplementary to grid searches for choosing neutralization proportions, and could be useful when performing feature neutralization with models that are known to be high/low variance. For example, when using random forests which are known to be variance reducing, you could use a lower neutralization proportion as the model is already lower variance. On the other hand, if you use neural networks which typically overfit due to their high variance, you might use a higher neutralization proportion to balance this out.

Background

Bias-Variance Decomposition

Given a target function and a chosen model to approximate it with, the out-of-sample mean-squared error (MSE) decomposes as the sum of a bias and a variance term. To understand the bias and variance of an estimator, it’s important to understand the concept of sampling an instance of a dataset from the data generating function, and fitting the model to this specific dataset. The bias term measures the distance of the average model (where the average is taken over datasets) from the true data generating function. The variance term measures how much each model trained on a specific instance of a dataset fluctuates around the average model as we change training datasets. Since the MSE is the sum of these two terms (plus a constant noise term), there is a tradeoff in that reducing the bias of an estimator will increase the variance and vice versa. This is why it is typically referred to as the bias-variance tradeoff.

Feature Neutralization

Feature neutralization is a technique that is commonly used in the Numerai tournament to reduce exposure to features that a model relies heavily on when making a prediction. Feature exposure is typically calculated as the correlation between each feature and a model’s predictions. Features that exhibit a high correlation with the predictions are those that the model is exposed to. In the event that these features stop being predictive of the target, the exposed model is likely to experience drawdowns and swings in performance.

Given a model trained on the training data, feature neutralization is performed during evaluation by making a prediction on the validation or test data, and fitting a linear model to those predictions with the exposed features. By taking the difference between the original predictions and an in-sample prediction from the feature neutralization linear model, the predictions are “neutralized”. The following pseudo code demonstrates the procedure:

Fit model

model.fit(X_train, y_train)

Predict on validation set

y_pred = model.predict(X_val)

Calculate feature exposures and select the most exposed features

exp_feas = calculate_feature_exposure(X_val, y_pred)

Neutralize the predictions (p is a neutralization proportion between 0 and 1)

pred_neutralized = y_pred - p*LinearRegression.fit(exp_feas, y_pred).predict(exp_feas)

Feature neutralization is done on a per-era basis. Depending on the size of the neutralization proportion, this process partially (or fully) removes the contributions of the exposed features from the original prediction. By doing so, the exposure of the model to these features is reduced.

Implementation Notes

In the forum and numerai_tools

library, you'll see feature neutralization implemented as something like $y_{\mathrm{pred}} - p X(X^TX)^{-1}X^Ty_{\mathrm{pred}}$

, where y_{pred}

is the vector of predictions from the main model, p

is the neutralization proportion and X

is the matrix of exposed features. The $(X^TX)^{-1}X^Ty_{\mathrm{pred}}$

part is just the classic solution to a $y = X\beta$

and $X(X^TX)^{-1}X^Ty_{\mathrm{pred}}$

is equivalent to LinearRegression.fit(exp_feas, y_pred).predict(exp_feas)

in the pseudo code. I think the implementation as it exists in numerai_tools

is quite unfortunate. In practice, you would almost never solve a linear equation by inverting X^TX

as this is slow, can lead to numerical instabilities and the inverse will not exist if the columns of X

are dependent. In practice, these kinds of equations are solved by optimization and I found a while ago that doing this made feature neutralization twice as fast. Maybe I'll fix this in the numerai_tools

library and submit a pull request…