Stacking Models for Improved Predictions

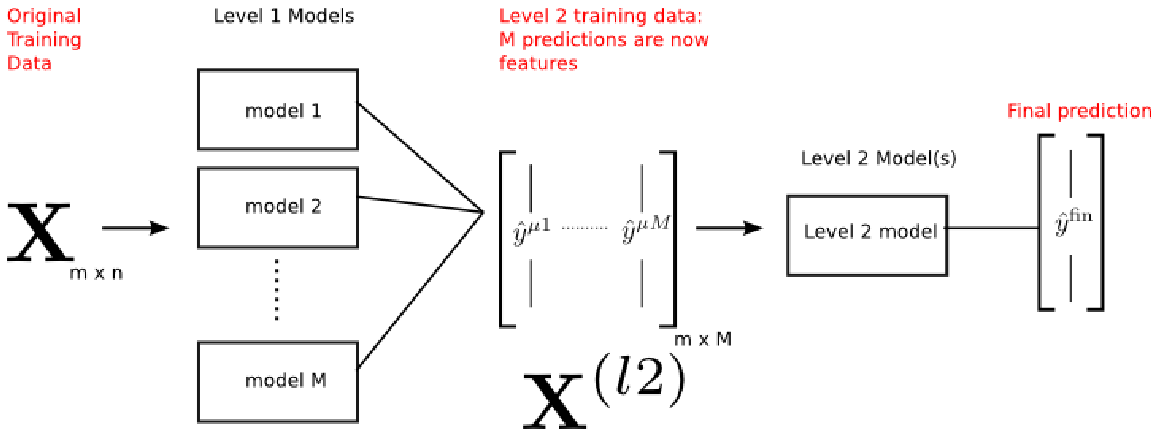
Tags: Ensemble Methods, Machine Learning, XGBoost

This post presents an example of regression model stacking, and proceeds by using XGBoost, Neural Networks, and Support Vector Regression to predict house prices.

By Burak Himmetoglu, UC Santa Barbara.

If you have ever competed in a Kaggle competition, you are probably familiar with the use of combining different predictive models for improved accuracy which will creep your score up in the leader board. While it is widely used, there are only a few resources that I am aware of where a clear description is available (One that I know of is here, and there is also a caret package extension for it). Therefore, I will try to workout a simple example here to illustrate how different models can be combined. The example I have chosen is the House Prices competition from Kaggle. This is a regression problem and given lots of features about houses, one is expected to predict their prices on a test set. I will use three different regression methods to create predictions (XGBoost, Neural Networks, and Support Vector Regression) and stack them up to produce a final prediction. I assume that the reader is familiar with R, Xgboost and caret packages, as well as support vector regression and neural networks.

The main idea of constructing a predictive model by combining different models can be schematically illustrated as below:



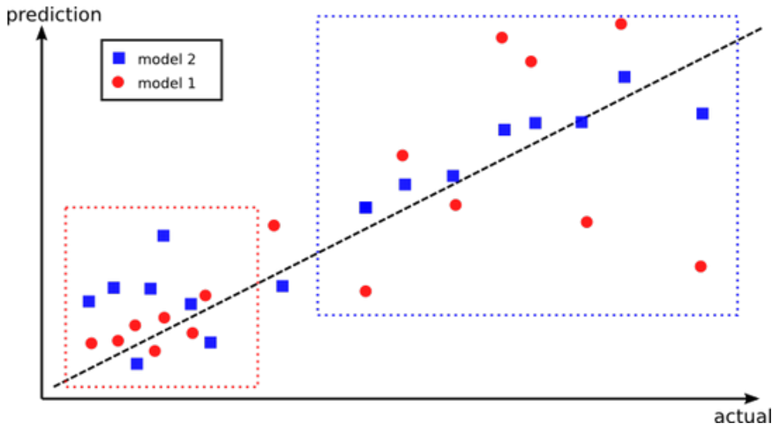
Let me describe the key points in the figure:

- Initial training data (**X**) has *m* observations, and *n* features (so it is *m x n*).
- There are M different models that are trained on X (by some method of training, like cross-validation) before hand.
- Each model provides predictions for the outcome (y) which are then cast into a second level training data (Xl2) which is now *m x M*. Namely, the M predictions become features for this second level data.
- A second level model (or models) can then be trained on this data to produce the final outcomes which will be used for predictions.

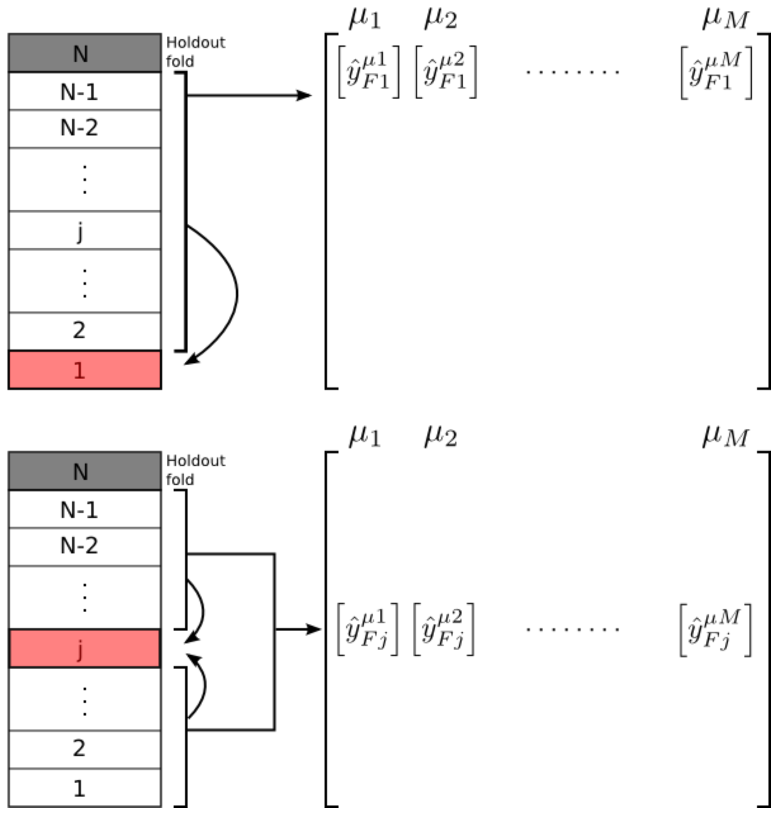
There are several ways that the second level data (Xl2) can be built. Here, I will discuss **stacking**, which works great for small or medium size data sets. Stacking uses a similar idea to k-folds cross validation to create **out-of-sample** predictions.

The key word here is **out-of-sample**, since if we were to use predictions from the M models that are **fit to all the training data**, then the second level model will be biased towards the best of M models. This will be of no use.

As an illustration of this point, let's say that **model 1** has lower training accuracy, than **model 2** on the training data. There may however be data points where **model 1** performs better, but for some reason it performs terribly on others (see figure below). Instead, **model 2** may have a better overall performance on all the data points, but it has worse performance on the very set of points where **model 1** is better. The idea is to combine these two models where they perform the best. This is why creating out-of-sample predictions have a higher chance of capturing distinct regions where each model performs the best.



First, let me describe what I mean by stacking. The idea is to divide the training set into several pieces like you would do in k-folds cross validation. For each fold, the rest of the folds are used to obtain a predictions using all the models 1…M. The best way to explain this is by the figure below:


N
N-1
N-2
j
2
1
Holdout fold
μ_1
μ_2
μ_M
$\left[\hat{y}_{F1}^{\mu 1}\right]$
$\left[\hat{y}_{F1}^{\mu 2}\right]$
$\left[\hat{y}_{F1}^{\mu M}\right]$
$\left[\hat{y}_{Fj}^{\mu 1}\right]$
$\left[\hat{y}_{Fj}^{\mu 2}\right]$
$\left[\hat{y}_{Fj}^{\mu M}\right]$

Here, we divide our training data into N folds, and hold the Nth fold out for validation (i.e. the holdout fold). Suppose we have M number of models (we will later use M=3). As the figure shows, prediction for each fold (Fj) is obtained from a fit using the rest of the folds and collected in an out-of-sample predictions matrix (Xoos). Namely, the level 2 training data Xl2 is Xoos. This is repeated for each of the models. The out-of -sample prediction matrix (Xoos) will then be used in a second level training (by some method of choice) to obtain the final predictions for all the data points. There are several points to note:

- We have not simply stacked the predictions on all the training data from the M models column-by-column to create a second level training data, due to the problem mentioned above (the fact that the second level training will simply choose the best of the M models).
- By using out-of-sample predictions, we still have a large data to train the second level model. We just need to train on Xoos and predict on the holdout fold (Nth). This is in contrast to model ensembles.

Now, each model (1…M) can be trained on the (N-1) folds and a prediction on the holdout fold (Nth) can be made. There is nothing new here. But what we do is that, using the second level model which is trained on Xoos, we will obtain predictions on the holdout data. We want that the **predictions from the second level training be better than each of the M predictions from the original models**. If not, we will have to restructure the way we combine models.

Let me illustrate what I just wrote with a concrete example. For the case of the House Prices data, I have used 10 folds of division of the training data. The first 9 is used for building Xoos, and 10th is the holdout data for validation. I trained three level 1 models:

XGBoost, neural network, support vector regression. For level 2, I used a linear elasticnet model (i.e. LASSO + Ridge regression). Below are the root-mean-squared errors (RMSE) of each of the models evaluated on the holdout fold:

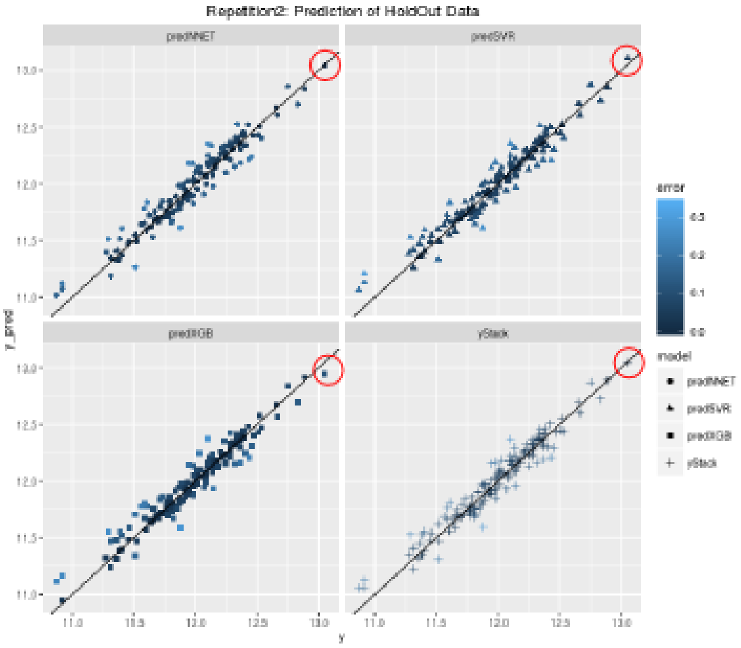
```
XGBoost:  0.10380062

Neural Network:  0.10147352

Support Vector Regression:  0.10726746

Stacked:  0.10005465
```

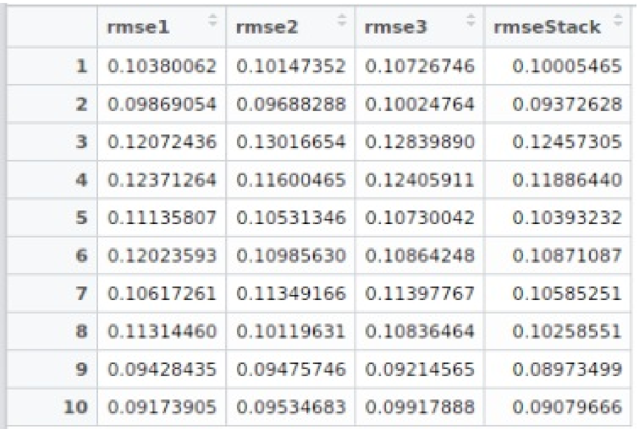
As clear from this data, the stacked model has slightly lower RMSE than the rest. This may look too small of a change, but when Kaggle leaderships are involved, such small differences matter a lot!



Graphically, once can see that the circled data point is a prediction which is worse in XGBoost (which is the best model when trained on all the training data), but neural network and support vector regression does better for that specific point. In the stacked model, that data point is placed close to where it is for neural network and support vector regression. Of course you can also see some cases where using just XGboost is better than stacking (like some of the lower lying points). However, the overall predictive accuracy of the stacked model is better.

One final complication that will further boost your score: If you have spare computational time, you can create repeated stacks. This will further reduce the variance of your predictions (something reminiscent of bagging).

For example, let's create a 10 folds stacking not just once, but 10 times! (say by caret's createMultiFolds function). This will give us multiple level 2 predictions, which can then be average over. For example, below are the RMSE values on the holdout data (rmse1: XGBoost, rmse2: Neural Network, rmse3: Support Vector Regression), for 20 different random 10-folds created. Averaging the final predictions from the level 2 predictions on these Xoos's (i.e. Xoos from stack1, Xoos from stack2, …, Xoos from stack10), would further improve your score.



	rmse1	rmse2	rmse3	rmseStack
1	0.10380062	0.10147352	0.10726746	0.10005465
2	0.09869054	0.09688288	0.10024764	0.09372628
3	0.12072436	0.13016654	0.12839890	0.12457305
4	0.12371264	0.11600465	0.12405911	0.11886440
5	0.11135807	0.10531346	0.10730042	0.10393232
6	0.12023593	0.10985630	0.10864248	0.10871087
7	0.10617261	0.11349166	0.11397767	0.10585251
8	0.11314460	0.10119631	0.10836464	0.10258551
9	0.09428435	0.09475746	0.09214565	0.08973499
10	0.09173905	0.09534683	0.09917888	0.09079666

Once we verify that stacking results in better predictions than each of the models, then we re-run the whole machinery once again, without keeping Nth fold as holdout data. We create Xoos from all the folds, and the the second level training uses Xoos to predict the test set which Kaggle provides us with. Hopefully, this will creep your score up in the leader board!.

Final word: You can find the scripts from my Github repo.

Note: If you have a classification problem, you can still use the same procedure to stack class probabilities.

Bio: Burak Himmetoglu is a Data Scientist and High Performance Computing (HPC) specialist with Ph.D. in physics. He has strong mathematical modeling, data analysis, and programming background, and is passionate about applying academic skills to solve difficult business problems and develop data products.

Original. Reposted with permission.

Related:

- Data Science Basics: An Introduction to Ensemble Learners
- Random Forests in Python
- Top R Packages for Machine Learning

<= Previous post
Next post =>

Top Stories Past 30 Days

Most Popular

1. **If I had to start learning Data Science again, how would I do it?**
2. **Must-read NLP and Deep Learning articles for Data Scientists**
3. **Know What Employers are Expecting for a Data Scientist Role in 2020**
4. **These Data Science Skills will be your Superpower**
5. **The List of Top 10 Lists in Data Science**
6. **Top Google AI, Machine Learning Tools for Everyone**
7. **Top Online Masters in Analytics, Business Analytics, Data Science – Updated**

Most Shared

1. **If I had to start learning Data Science again, how would I do it?**
2. **Must-read NLP and Deep Learning articles for Data Scientists**
3. **The List of Top 10 Lists in Data Science**
4. **Top Google AI, Machine Learning Tools for Everyone**
5. **Top Online Masters in Analytics, Business Analytics, Data Science – Updated**
6. **5 Different Ways to Load Data in Python**
7. **How to Evaluate the Performance of Your Machine Learning Model**

KDnuggets Home » News » 2017 » Feb » Tutorials, Overviews » Stacking Models for Improved Predictions (17:n07)