Economics 1660: Big Data Prof. Daniel Björkegren

PS 5: Prediction II

If we tightly fit a model to training data, it can perform poorly out of sample. How can we train models to get strong performance on out-of-sample data?

We will continue working with data on loans granted by Kiva. We've constructed a large dataset which is split into three subsets: A, B, and C. In PS4, we gave you the full data for subset A, and everything but the labels (outcomes) for subset B, and asked you to turn in predictions for B. In this problem set we'll give you the full data for both subsets A and B, and ask you to develop methods that fit well out of sample. At the end, we'll give you unlabeled data for subset C and ask you to turn in predictions for C.

*Follow the instructions presented below and **answer the questions in bold.***

1. Train / Test

Use Python to read in the data from the file *loans_AB_labeled.csv*. Using the model you created in PS4, predict the *days_until_funded* variable for each loan. Compare each loan's predicted value versus the actual value (given by the *days_until_funded* variable in the dataset).

a. **What is the MSE of the actual and predicted values**
b. **How does the value in (1) compare to the accuracy you achieved within the sample A? Why do you think you observe this difference?**

If you achieved especially high accuracy within your sample, you probably noticed that the model did not perform as well when tested on the out-of-sample data. This is problematic – the criteria we use to evaluate models determine the model we end up with, and this result suggests that the most straightforward criteria leads to models that don't work well.

Let's revisit what we did in PS4, and see if we can do better. We'll go back to using subset A.

First, we need a better way to evaluate models. Let's split the data (in this case *loans_A_labeled.csv*) into two subsets, at random—let's call them A1 and A2—and train the model using the data from A1, while holding out A2 for evaluating the model.

Now that we have this improved evaluation criterion, we can change our method to still train on A1 but obtain better performance on A2. Start by using your method from PS4 to build a model based on the data in A1, and test the accuracy (MSE) on A2. Then, think about what drives the gap between performance on the training sample and test sample. Experiment with modifications to the method to improve out of sample performance.

For example, you can try:

- Changing the depth of the tree
- Expanding or limiting the variables you generate by some criteria
- Pruning the resulting tree

Try several modifications and report:

c. What was the lowest MSE you were able to achieve on A2? Describe that model briefly in terms of what variables you included, number of levels, and any other important decisions you made.

d. In general, what types of tweaks improved your accuracy on A2? What changes decreased your accuracy?

2. Random Forests

Given how closely decision trees can fit themselves to their training data, it's not surprising that they have a tendency to overfit. One way of avoiding this is a technique called *random forests*. The idea is to build multiple distinct decision trees and average their predictions.

Random forests are one of the most popular and versatile classification/regression models around. We will build a basic one in this part of the assignment. A random forest is made up of many different random trees. The randomness of these trees comes from two modifications to the tree-building process we have used so far:

- *Bootstrapping*: A bootstrap sample is the result of randomly choosing a number of elements of a set *with replacement*. For example, from the data {A, B, C, D, E}, a bootstrap sample of size 3 could be {D, B, A}, {A, A, C}, or {C, D, C}. When building the trees for a random forest, rather than training each tree on all the inputs in the training set, we train each tree on a bootstrap sample of the inputs.
- *Random Split Candidates*: Every time we make a split in a decision tree for a random forest, rather than looking at all the remaining attributes, we first choose a random subset of them and then split on whichever of those is best.

Once we have constructed a random forest (represented as a list of random trees), we make predictions based on the average prediction made by each tree in the forest.

Your task:

You will now use random forests to try to improve your prediction accuracy. The most difficult parts have been done for you. The file *forest_support.py* contains three support functions:

- bootstrap_sample(inputs, length) takes a list of *inputs* from the training data and a *length*, and it returns a bootstrap sample (in the form of a list) of that length from the list of inputs. **Your inputs should be a collection of tuples (*a*, *b*) where *a* is the feature dictionary and *b* is the label (*days_until_funded*) for each loan.**
- build_forest_tree(inputs, num_levels, num_split_candidates) takes a list of *inputs* (in the form of tuples), the number of levels (*num_levels*) to use in building a tree, and the number of split candidates (*num_split_candidates*) to randomly choose at each split in the tree. The function returns a tree that can be used as one of the trees in your random forest.
- forest_predict(trees, loan) takes a list of *trees* (your random forest) and a *loan* (in the form of a tuple) and returns the average prediction for that loan

There is no need to even open *forest_support.py*, but **make sure you include the following line at the top of your code in order to access the support functions:**

```
from forest_support import *
```

Your task is to use the support functions to build a random forest (list of trees) using the loans from *loans_A_labeled.csv* as your training data. Since you have already loaded the data from the CSV file, simply use the *build_forest_tree* function to generate several trees from bootstrap samples of the loans. To start out, use any values you want for the parameters of the support functions, but so that your code does not take outrageously long to run, we recommend limiting the number of trees in your random forest to 1000. (Building the forest should only take a few lines of code if you use the support functions.)

Now that you have a random forest, we can use it to make predictions on our test data, the loans from *loans_AB_labeled.csv*.

e. Use the forest_predict support function to predict the *days_until_funded* value of each of the loans in *loans_AB_labeled.csv*. Compare the predictions with the actual values. What is the MSE? What parameters did you choose when calling the support functions? How many trees did you use to build your random forest?

f. Adjust the parameters and number of trees in your forest and observe how the prediction accuracy changes. Write a few sentences about the effects of some of your changes. What types of tweaks in the parameters would you expect to improve out-of-sample accuracy? Why?

3. Turn in Out of Sample Predictions

Once you've developed a method that you are confident in, build a model using the loans from *loans_AB_labeled.csv*. You can choose any method we have explored – whichever you found to be the most accurate on out-of-sample data.

You will now use your model to make predictions on a new, unlabeled dataset, "loans_C_unlabeled.csv". This dataset is identical in format to loans_AB_labeled.csv, but the classification variable *days_until_funded* is omitted. Use your model to predict the value of *days_until_funded* for each loan in loans_C_unlabeled.csv. Save your predictions in the form of a CSV file. The first column should be the ID of the loan, and the second column should be your prediction. In the header row, name the first variable "ID", and the second variable "days_until_funded" followed by the first and last initials of your group members. For example, Professor Björkegren and Simon Freyaldenhoven's group would turn in results of the form:

ID, days_until_funded_DB_SF
1,5
2,0
3,8
…

Save these as *loans_C_predicted_[*Group Initials Here*].csv* and turn this in on Canvas.