Carolina Carvalho Manhães Leite (leite2)
IE598 MLF F19
Module 6 Homework (Cross validation)

Using the ccdefault dataset, with 90% for training and 10% for test (stratified sampling) and the decision tree model that you did in Module 2:

Part 1: Random test train splits

Run in-sample and out-of-sample accuracy scores for 10 different samples by changing random_state from 1 to 10 in sequence. Display the individual scores, then calculate the mean and standard deviation on the set of scores. Report in a table format.

Part 2: Cross validation

Now rerun your model using cross_val_scores with k-fold CV (k=10). Report the individual fold accuracy scores, the mean CV score and the standard deviation of the fold scores. Now run the out-of-sample accuracy score. Report in a table format.

Part 3: Conclusions

Write a short paragraph summarizing your findings. Which method of measuring accuracy provides the best estimate of how a model will do against unseen data? Which one is more efficient to run?

Part 4: Appendix

Link to Github repo

I utilized some pre-processing methods, trying to balance some behaviors found in the dataset:

- Unbalanced target classes (23,364 non-defaults and only 6,636 defaults);
- Numeric attributes with different ranges (we have age and currency amounts coexisting);
- Categorical attributes

For the numeric features I standardized all the columns and imputed the median of the non-missing values into the missing ones. For the categorical values, I utilized the "OneHotEncoder" in order to transform the categories to dummy variables (so that we can use the DecisionTreeClassifier to model our data). All of those transformations were implemented using the "Pipeline" method. The unbalance in the target value was addressed by a stratified sample (when splitting the train and test datasets) and also by a Stratified K-Fold Cross Validation method embedded in the "cross_val_score" method (Part 2 of the assignment).

For the DecisionTreeClassifier I kept the both parameters (max_depth and random_state) as constants (7 and 1, respectively). I could've changed the max_depth – a hyperparameter optimization approach – but I kept it as a constant for us to compare the 2 methods of splitting train/test datasets. Also, I didn't change

the value of random_state because, for Part 1, this value was already supposed to change (when building the train/test samples). Again, I maintained this parameter as a constant for the classifier, so that the only change between parts 1 and 2 is the sampling method.

Part 1: Random test-train splits

Results for the out-of-sample dataset:

Random_state	1	2	3	4	5	6	7	8	9	10
Acc scores	0.820	0.822	0.815	0.822	0.819	0.816	0.825	0.805	0.814	0.815
Acc scores (mean)	0.817									
Acc scores (std)	0.006									
Process time	20.642									

The results for the in-sample dataset can be seen in the execution LOG of the code.

Part 2: Cross validation

Results for the 10 folds:

Fold	1	2	3	4	5	6	7	8	9	10
Acc scores	0.810	0.821	0.814	0.810	0.819	0.814	0.814	0.820	0.824	0.827
Acc scores (mean)	0.817									
Acc scores (std)	0.005									
Process time	14.953									

Part 3: Conclusions

The results for the two validation methods were practically the same, with a slightly difference in the standard deviation – it is lower for the K-Fold CV, which may suggest this method is more reliable for us to analyze the performance of a model. After seeing the results, I even started varying the random_state of the classifier itself (in addition to the varying random_state of the "train_test_split"). It didn't change anything. I suppose the pre-processing treatments and the fact that both samples were stratified may have led to a more "stable/boring" dataset, with stable models and little "surprises".

One difference we can see, though, is in the process time for both methods. When we used the K-Fold CV method (already implemented in Python inside one of their packages), the process took 72% of the time to be compiled when compared to our random/manual test-train splitting, which shows that an already implemented/established algorithm is more efficient to run.

Part 4: Appendix

https://github.com/leiteccml/Carolina_2019/tree/master/IE598_F19_HW6