Carolina Carvalho Manhães Leite (leite2)
IE598 MLF F19
10/11/2019
Module 7 Homework (Random Forest)

Using the ccdefault dataset, and 10-fold cross validation described in Raschka;

Part 1: Random forest estimators

Fit a random forest model, try several different values for N_estimators, report in-sample accuracies.

Using the same dataset from last week (ccdefault.csv), I created the training and test samples (test_size = 0.1) and stratified the sample (in y) because, as we already saw in the last assignment, the target variable is unbalanced. Since we're using a Random Forest model, I didn't perform any pre-processing treatments (standardization, imputation of missing values, creation of dummy variables etc): the Random Forest technique is an ensemble of decision trees, which can handle data in their original range of values, as well as categorical or missing entries (it'll just consider them as another category). Actually, last week I didn't necessarily need to apply all of those processes (but it was a good thing to do for me to use the pipeline feature, anyway).

After instantiating the Random Forest, I used the GridSearchCV approach (with 10 K-Fold CV samples) to perform the hyperparameter tuning, setting 5, 10, 15, 20, 25, 50, 100 and 125 as the number of estimators (i.e. number of trees). The accuracy score will be used as metric for me to choose the best model (i.e., the number of estimators that led to the higher accuracy score in the test sample).

We can get some interesting insights from the GridSearchCV approach: the mean and standard deviation of the accuracy scores in both the training and the testing samples, as well as the time (in seconds) needed to fit the model and score it. Those results can be seen in the following table:

Table 1. Results for a Random Forest model as the number of estimators get higher.

N_estimators	Mean acc. score (train)	Std dev acc. score (train)	Mean acc. score (test)	Std dev acc. score (test)	Fit time	Score time
5	0.970	0.001	0.787	0.014	0.562	0.009
10	0.979	0.001	0.805	0.013	1.103	0.014
15	0.992	0.001	0.808	0.011	1.648	0.020
20	0.993	0.001	0.812	0.012	2.235	0.026
25	0.996	0.000	0.813	0.012	2.749	0.032
50	0.999	0.000	0.816	0.015	5.517	0.061
100	0.999	0.000	0.816	0.013	10.876	0.121
125	0.999	0.000	0.816	0.013	13.045	0.137

Graphically:

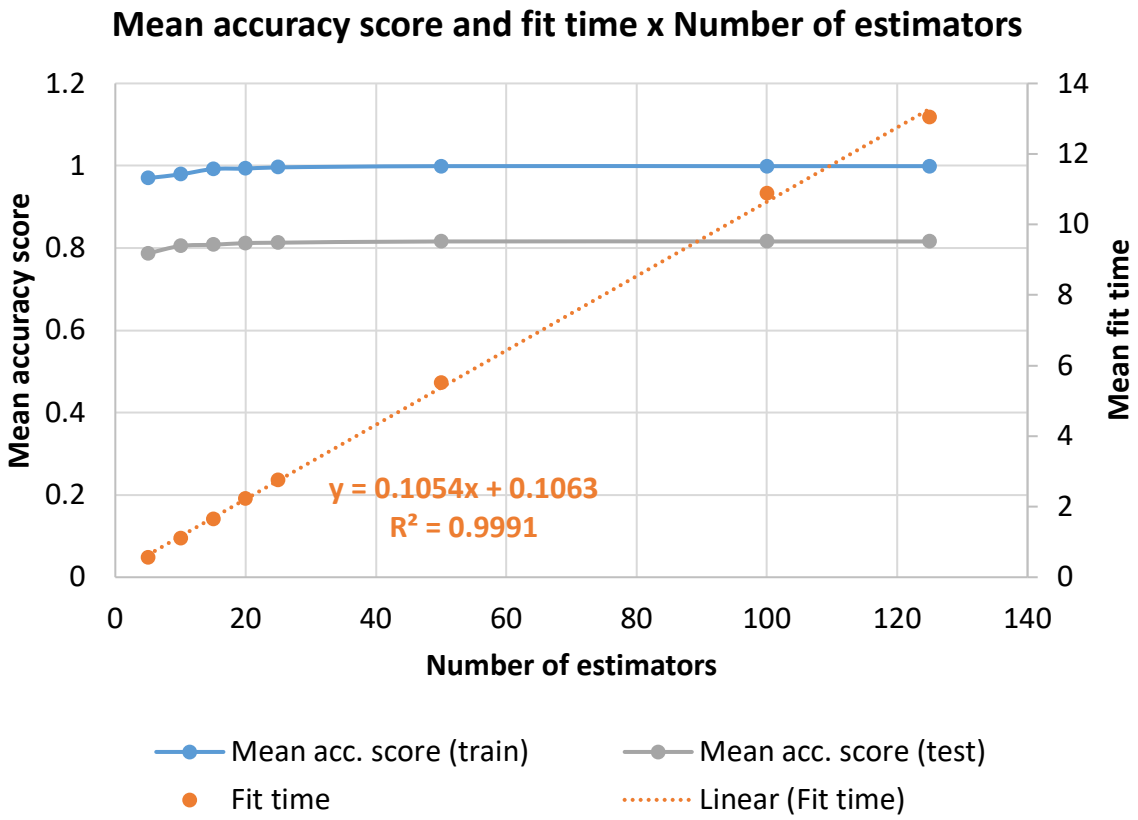

Figure 1. Chart with the mean accuracy score and fit time for different values of "N_estimators".

Part 2: Random forest feature importance

Display the individual feature importance of your best model in Part 1 above using the code presented in Chapter 4 on page 136. {importances=forest.feature_importances_}

Choosing the model with 50 estimators as the optimum one (I'll justify this choice later in the report), I was able to construct a bar chart with the features' importances:

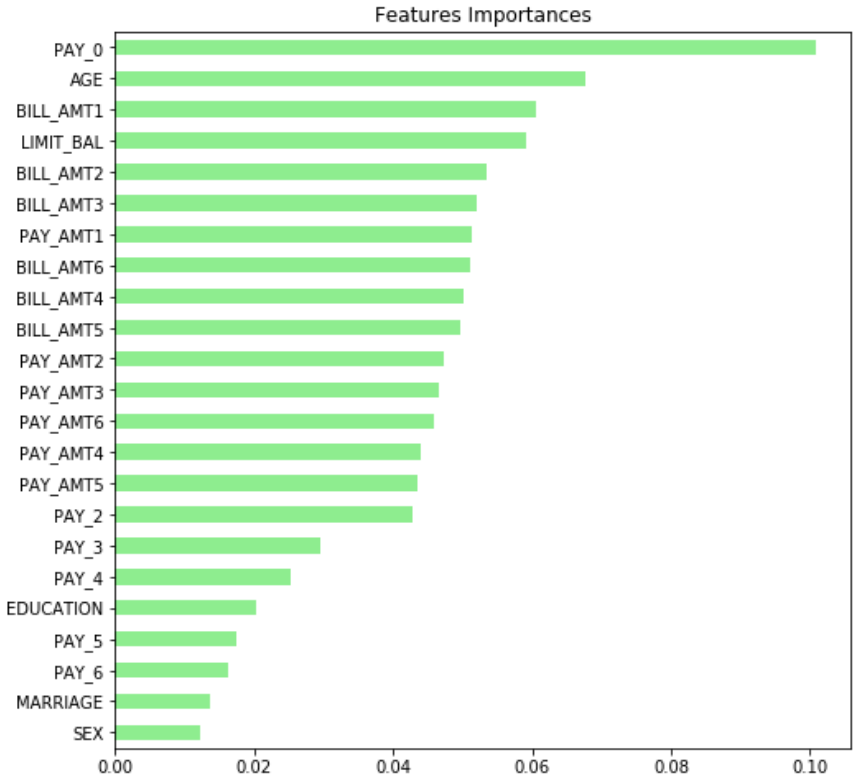

Figure 2. Bar chart with the features' importances for a model with 50 estimators.

Part 3: Conclusions

Write a short paragraph summarizing your findings. Answer the following questions:

a) What is the relationship between n_estimators, in-sample CV accuracy and computation time?
b) What is the optimal number of estimators for your forest?
c) Which features contribute the most importance in your model according to scikit-learn function?
d) What is feature importance and how is it calculated? (If you are not sure, refer to the Scikit-Learn.org documentation.)

We can take some interesting conclusions from the results above. As the number of estimators increases (in our case, since it's a Random Forest ensemble model, estimators = # trees), both the train and the test mean accuracy scores get higher and the standard deviations decrease, but only up to a "saturation" level. We'd expect from the very high accuracy training score (as high as 0.999) that this would already represent a case of overfitting, being accompanied by a decrease in the testing accuracy scores. That's not what we observe, though. Both train and test accuracies show an upward trend up until stability, and from this point on **both** curves remain stable at a certain level. This seems to corroborate the theory that the Random Forest method, being an ensemble of many simple/basic individual trees (with depths as low as

1), doesn't lead to overfitting (exactly because of the restriction imposed to the depth of the individual trees).

This behavior also shows us that, from the stability point on, it isn't worth to increase the number of estimators, since the gain in accuracy is barely marginal. More than that, if we take into account the fitting time (also plotted in the chart), we can see it has a positive linear relationship with the number of estimators. This makes our decision a little more nuanced. For example, let's compare the results below:

Table 2. Results for a Random Forest model as the number of estimators get higher.

N_estimators	Mean acc. score (train)	Std dev acc. score (train)	Mean acc. score (test)	Std dev acc. score (test)	Fit time	Score time
25	0.996	0.000	0.813	0.012	2.749	0.032
50	0.999	0.000	0.816	0.015	5.517	0.061
100	0.999	0.000	0.816	0.013	10.876	0.121
125	0.999	0.000	0.816	0.013	13.045	0.137

We can see that, for 50 estimators and more, the accuracy score reaches a peak value and remains stable for both the train and test sets. One could argue that the optimum choice for "N_estimators" should be 100, since we have a slight drop in the standard deviation for the test set (which is a good thing), but this happens at expense of the fitting time (almost twice as high). If we take into account all of those factors, **the best value for "N_estimators" should be 50**.

Establishing the best model as the one with 50 estimators, we can extract the features' importances, displayed above in the **Figure 2**. **Table 3** shows the same results:

Table 3. Variables and their importances in the model.

Variable	Importance (%)	Cumulative Importance (%)
PAY_0	10.07%	10.07%
AGE	6.78%	16.85%
BILL_AMT1	6.05%	22.90%
LIMIT_BAL	5.91%	28.81%
BILL_AMT2	5.34%	34.14%
BILL_AMT3	5.21%	39.36%
PAY_AMT1	5.13%	44.49%
BILL_AMT6	5.12%	49.61%
BILL_AMT4	5.02%	54.62%
BILL_AMT5	4.97%	59.59%
PAY_AMT2	4.73%	64.32%
PAY_AMT3	4.65%	68.97%

PAY_AMT6	4.58%	73.56%
PAY_AMT4	4.40%	77.96%
PAY_AMT5	4.34%	82.30%
PAY_2	4.28%	86.58%
PAY_3	2.95%	89.53%
PAY_4	2.52%	92.04%
EDUCATION	2.03%	94.08%
PAY_5	1.74%	95.81%
PAY_6	1.62%	97.43%
MARRIAGE	1.35%	98.79%
SEX	1.21%	100.00%

We can see that, even though "PAY_0" – with 10% – is the most important variable, all 23 variables have some degree of importance in this Random Forest model. In Scikit-learn, the features' importances are accessed through the attribute "feature_importance_", implemented using the concept of "mean decrease impurity": the total decrease in node impurity (weighted by the probability of reaching that node – approximated by the proportion of samples reaching the node) averaged over all trees of the ensemble. Then, the importances are normalized: each feature importance is divided by the total sum of importances, in a way that they all add up to 1 and describe how much a single feature contributes to the tree's total impurity reduction.

Part 4: Appendix

https://github.com/leiteccml/Carolina_2019/tree/master/IE598_F19_HW7