In some industries, model explan

Given a scenario, you have a model built, are you done?

Unfortunately, It is not end of your data science project, it is new start of many other things.

We will need to validate a few things so that we know model is good.

1. Besides good performance metrics score, what are the import features which support prediction?
2. It would be good to check visually if model is overfitting and underfitting
3. If it is imbalance data, what threshold is optimized to choose from
4. Other useful chart to check model performance

It is easier to explain with an example, so let us use the previous project to explore the question we listed. For detail of previous project, I include the link here.

First, I need to import the pickle file I saved from the previous project. In order to do that, I need to import all packages and custom classes, functions as I know it will run into error without those.

In cell 8, I imported sklearn.model\_selection object.

Feature importance

In some industries, model explainability is crucial. Imaging we built an insurance pricing model, but we cannot explain how model works. When we submit the model to state regulator for approve. We will fail that because regulator need to know why we increase insurance premium. Over the past 20 years, there are a few important machine learning framework. Depending on different model, scikit learn generally provide model property to reveal that; whereas deep learning framework is famous for its blackbox due to hidden layer despite of fairly high performance.

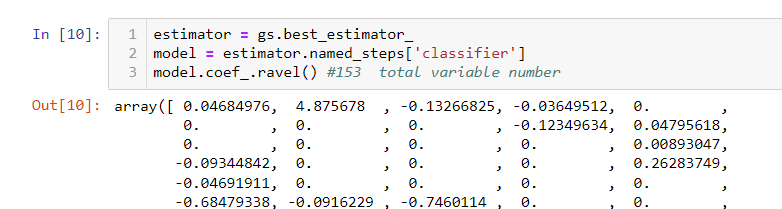
Since I used scikit learn in previous blog, I will more focus on some of the ways we can use scikit learn feature.

In Dr. Bronlee’s blog, he illustrated three importance ways to get feature importance.

https://machinelearningmastery.com/calculate-feature-importance-with-python/

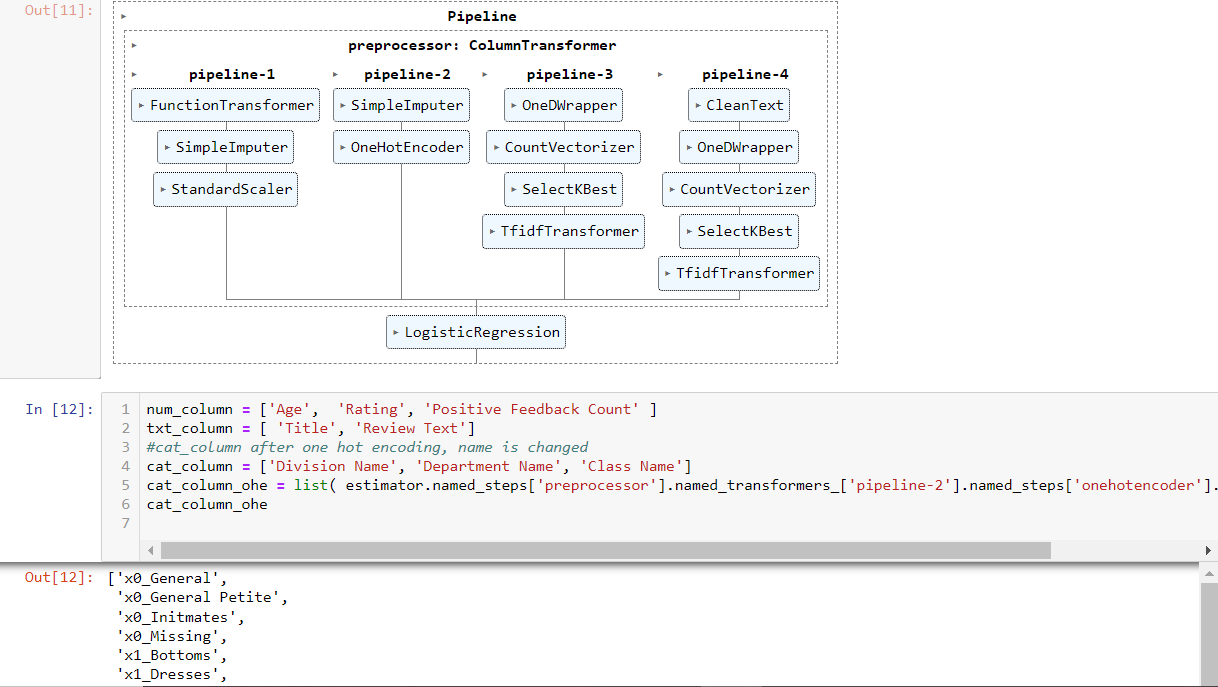
1. Coefficients as feature importance if it is linear model
2. Tree based feature importance (Decision tree, Random Forrest, XGBoost et al )
3. Permutation feature importance, (Pass scrambled predictor to model to check performance drop to get importance).

The optimized model is logistic regression. It is the linear model, let us use the first method, i.e. coefficient method. We can use name steps to retrieve model direct from sklearn.model\_selection object. Use its property coef\_. We can see 153 variable. These variable value indicate how much impact this variable changes. Please note, your data need to be scaled to similar level. In my case, all value is between (-1, 1). So the impact between each variable is relative comparable.



With coeffient, we will need to map variable (feature) to it.

In my previous blog, I have numeric features, categorical features and text features.



Num\_column, categorical\_column, Title in txt\_coloumn and ‘Review Text’ in txt column passed through pipeline-1 ~pipeline-4 with in the column transformer respectively. After the transformation, the results are usually in the sparse matrix format. Unlike dataframe, this format does not have columns. So in order to know the feature importance, we will have to retrieve the feature name if they went through something like one hot encoding, which will expand one column to multiple columns.

Let us see how we get feature name for the sparse matrix.

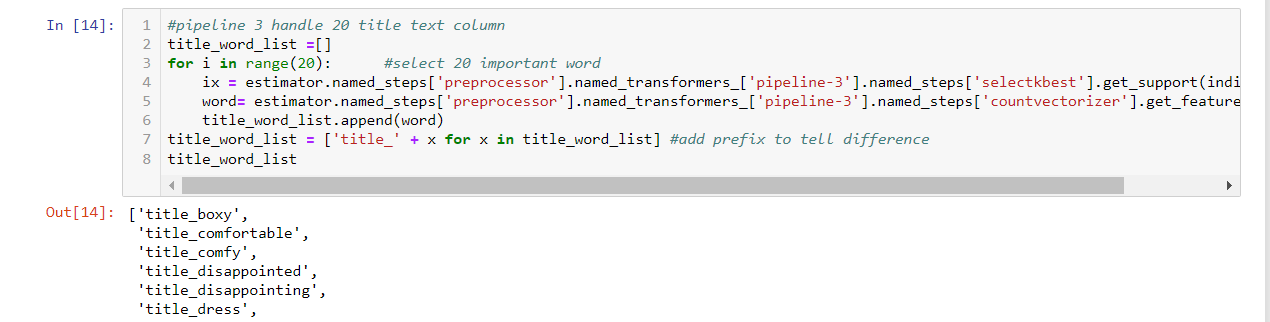
First of all, they follows sequence, i.e., it always process data from left to right (pipeline 1 to pipeline 4)

Num\_column: just simple imputing and scaling. The column number has not changed.

Categorical column: it has column expanding transformer, onehotEncoder. Luckily, sci-kit learn provide function such as named\_steps, named\_transformers (see cell 12 line 5) so that you can navigate to the onehotencoder step. Then you can use get\_feature\_names() to get all feature names (not show in the screenshot, too wide). Notice here, scikit learn rename the three variables (Division, Department, Class) to X0-X2.

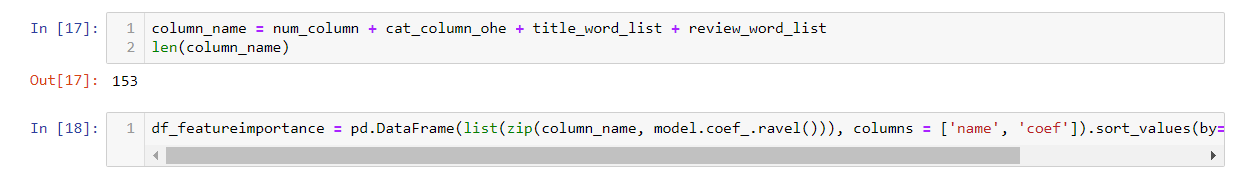
By the way, showing pipeline like cell 11 is helpful for you to navigate complicate transformation

Text column, I divide Title and “Review Text” into two pipeline (3 and 4). Text is tokenized into words (column expanding). We select 20 and 100 important words respectively with selectKBest transformer (column number changed too). These Text value is later used as column name.

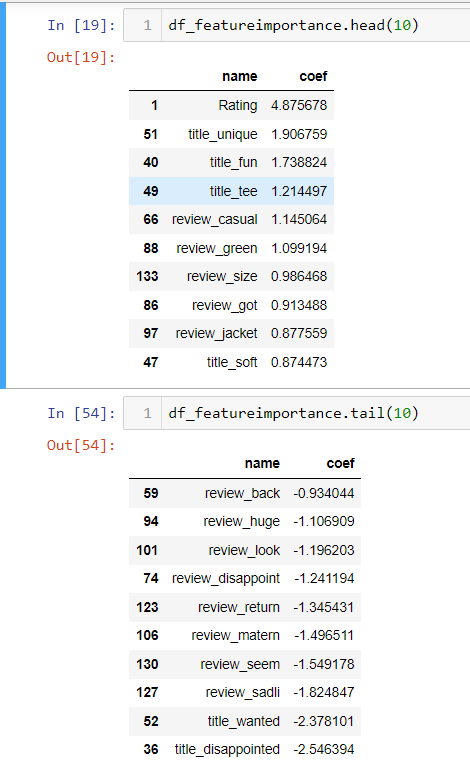


Here we can first get word index from selectKbest step. Then use the index to get actual word from previous step countvectorizer. There are 20 in Title, so I loop 20 time and rename the column to add prefix ‘title’ to distinguish it from ‘review’ . For Review Text, it works similarly. But just change 20 to 100.

Next, I piece together all the feature name together and combine it with coefficient.



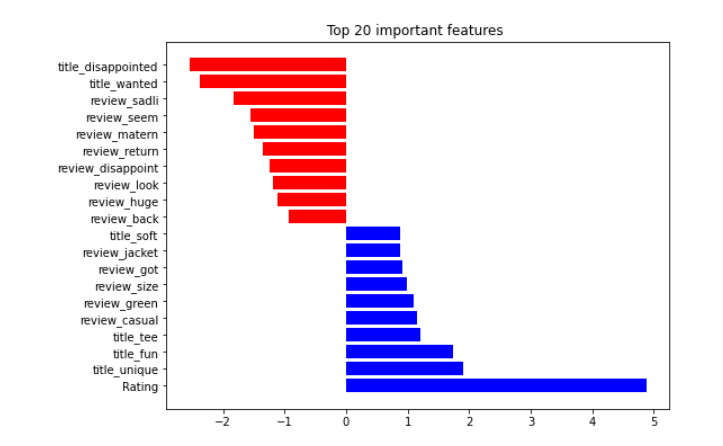
Let us see what those important player are.



I listed top 10 and bottom 10 important features. Notice bottom 10 is also important. It just negatively impact the prediction towards negative target (not recommended, target =0)

By reviewing those features, it makes much sense to me. If rating high definitely positively correlated with recommended (target =1), I don’t see categorical feature play important role here. Probably those are neutral. Others, in top 10 features are mainly good words. Bottom 10 features are mainly bad words. Only question here is title\_wanted in bottom 10, meaning word “wanted” in the title. I would think this is a positive word. But maybe, I don’t understand fashion. In fashion world, if other people want, it might not be a good thing. But it worth taking time to see what original context to see if this is the case and make further adjustment.

If I show this in the plot



These analysis are all at the detailed level, I often ask myself. What if we treat expanded categorical/text feature into one feature. As a whole, what feature importance landscape would be?

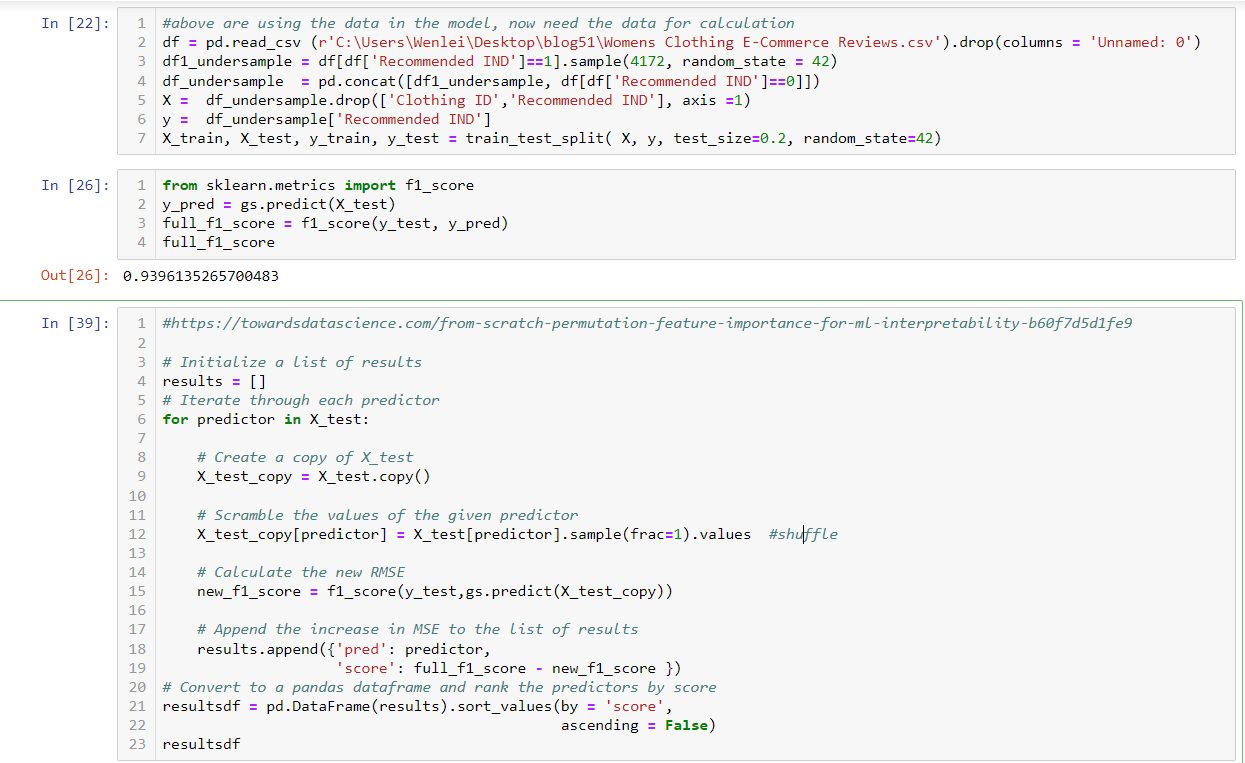
For that, we can use permutation feature importance calculation. Idea is as follows.

We already had a model and we knew performance would be.

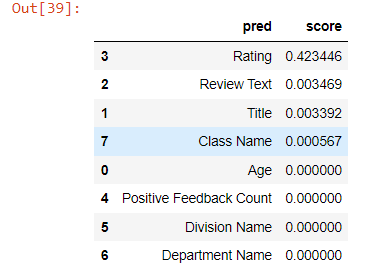
Now if I scramble/shuffle a feature value and pass the dataset to the model, you will see performance drop. If that particular feature is important, the drop is higher.

The following is the process. I modify part of code from this blog.

<https://towardsdatascience.com/from-scratch-permutation-feature-importance-for-ml-interpretability-b60f7d5d1fe9>



Before this section, I don’t need source data, because all model related data is pickled and you can retrieve from pickle. Since I need to do some data scrambling. I reimport data and did a prediction at cell 26 and use f1 as performance metrics. We get without scrambling is 0.94. now in cell 39, we loop through each column, in row 12, we shuffle the record. We recalculate the f1 score and compare it with baseline at row 19. Then we put all change into a dataframe and sort it



From the result, still you will see Rating is most important feature. Two text feature are still rank 2 and 3, but value wise is not as important as the detail level. This make sense, because it contain both positive and negative word, which might balance the impact as a whole.

There is a popular feature importance package called Shap. The difference between permuation importance and Shap is: the formed determined by performance metrix drop, while the latter is magnitude of feature attributions. Shap can also explain deep learning model. So check it out

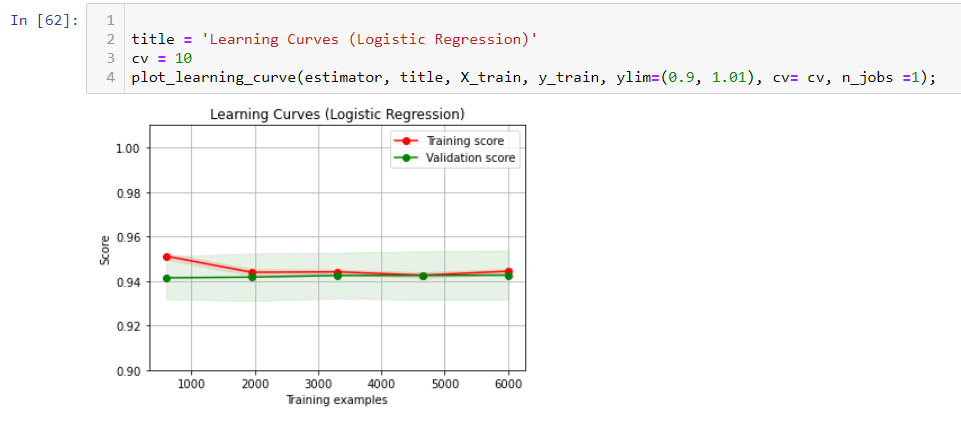
https://shap.readthedocs.io/en/latest/example\_notebooks/overviews/An%20introduction%20to%20explainable%20AI%20with%20Shapley%20values.html

Generally, if you see your model perform well with your test dataset. It is good sign. But it is reassuring for you that if you have a chart to show the learning curve.

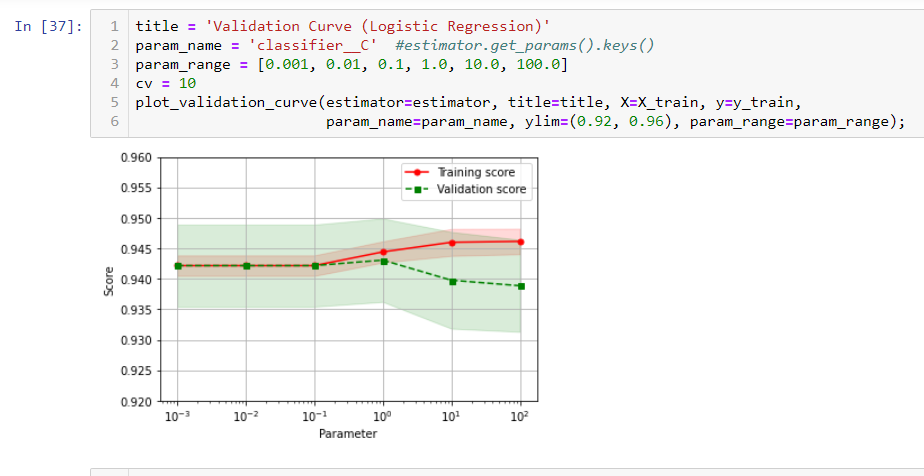
In Marcelino blog, he has two functions which I think it is useful.

<https://www.kaggle.com/code/pmarcelino/data-analysis-and-feature-extraction-with-python/notebook>

The function can be found in the notebook



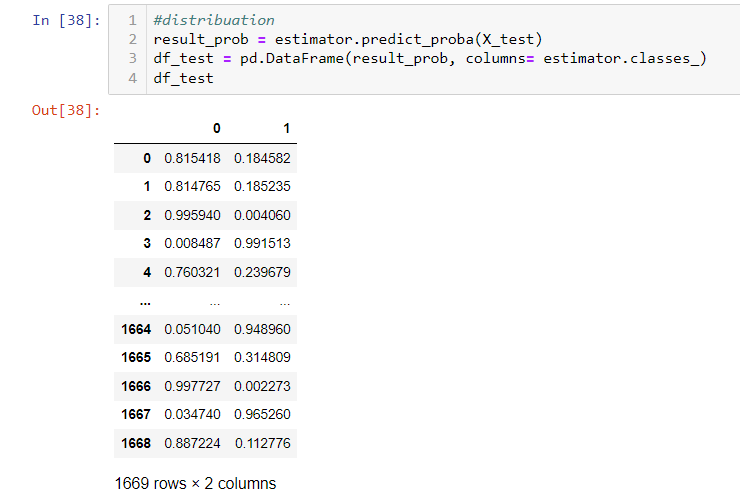
Basically, if your learning score is high, there is no underfitting. If you don’t see obvious space between two curve, there is no overfitting. In my case, I don’t see obvious underfitting and overfitting.



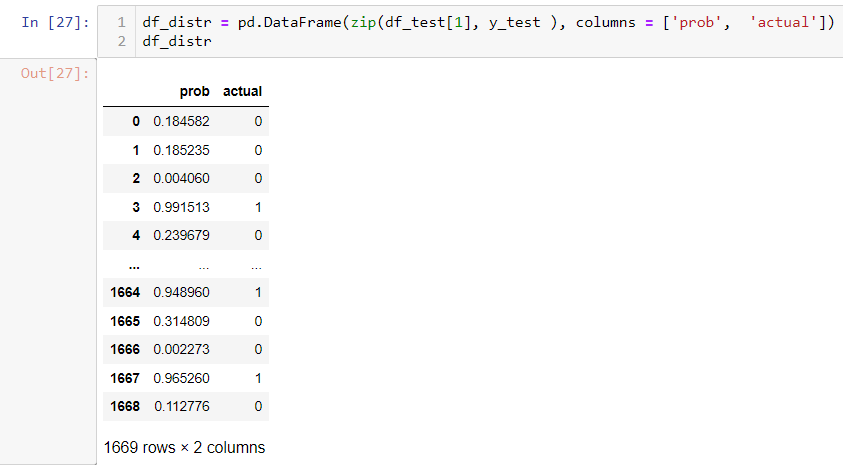
The other function is use to check hyperparam. You can see at 10e-1, the two curve start separate. Looks hyperparam C should be choose 10e-1 or lower. This is consistent with the best param in the previous blog.

Choose optimize the threshold.

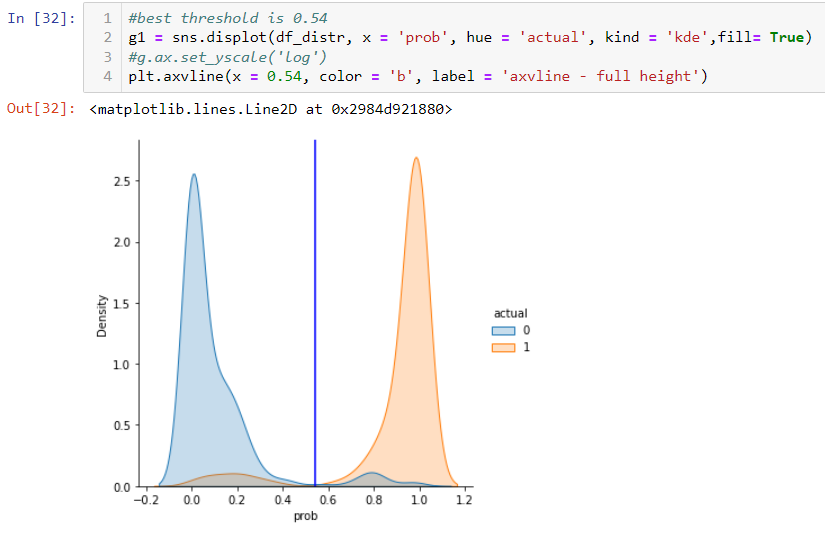
When doing classification, the algorithm gives a possibility for each record, then the final label is assigned by comparing the possibility with the threshold. By default, the threshold is 0.5. But it not always the case. I have seen the optimized threshold at 0.05 for some imbalanced dataset. How do you find the optimized threshold out. The idea is you put together your predicted probability with your target and plot it and find the optimized threshold.



First, you use predit\_prob function to get the probability.



second, you list possibility with the actual label.



Then you can use displot in searborn to plot the distribution. You can find where the optimized threshold point. In this particular case, it is close to 0.54

1. Other useful chart to check model performance

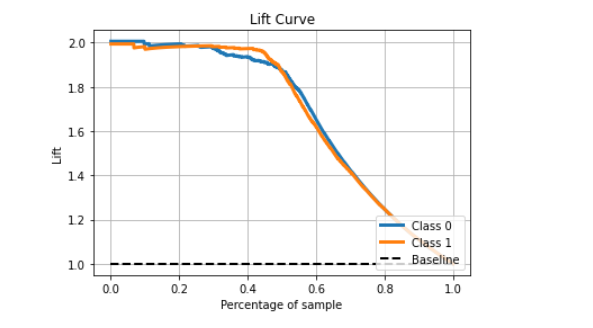
You can draw a ROC curve to see how good the performance is. You can use AUC to compare different run.

The list chart can tell you how you model can improve predict comparing without model. In this case, it improve performance by about 2 fold when you compare top 40% sample.

To understand the lift chart, the following link will be helpful.

<https://scikit-plot.readthedocs.io/en/stable/metrics.html>

<http://www2.cs.uregina.ca/~dbd/cs831/notes/lift\_chart/lift\_chart.html >



Thanks for following along.

It is a long series. The notebook is here.

Keep safe.

Wenlei