MODULE 09 - Piscine Python for Data Science

Machine Learning: Advanced

Summary: Today we will help you master advanced tasks involved in machine learning in Python.

Contents

Chapter I

Foreword

There is real data science and data science for competitions. The difference is the same as between a porter and weightlifter. A porter can be bad at heavy lifting competitions and a weightlifter can underperform on a real job. The same is true regarding data science: in companies, you should have a broader skill set including, for example, soft skills. You need to understand the business, and to be focused on profit or the organization's other goals. But in competitions, you simply need to be really good at achieving the competition's target metrics. You may create a super heavy machine learning model that can be 0.00001 better than the model of your next competitor and nobody cares how long it takes to make predictions, how interpretable the model is, or how many computational resources it requires. In business, of course, all these criteria matter.

Nevertheless, you may use competitions for improving your skills. The most popular platform is Kaggle. There are lots of public datasets and competitions. You may even try to win a prize, but most likely, you will only earn experience, which is not bad at all. It can be good for building your portfolio. You may organize a team and learn from each other. You can improve your collaboration skills. The platform allows you to pursue the machine learning track as well as the data exploration track: whatever you find more attractive.

So think about it as a great tool for continued growing in the field, but keep in mind that there are lots of things that are needed in real life besides building machine learning models and optimizaing hyperparameters.

Chapter II

Instructions

- Use this page as your only reference. Do not listen to any rumors or speculations about how to prepare your solution.
- Here and further on we use Python 3 as the only correct version of Python.
- The python files for python exercises (module01, module02, module03) must have the following block in the end: if ___name___ == '___main___'.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. If that fails, try your neighbor on the left.
- Your reference material: mates / Internet / Google.
- You can ask questions in Slack.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!

Chapter III

Specific instructions of the day

- Use Jupyter Notebook to work with your code
- For each major subtask in the list of any exercise (black bullets), your ipynb file should have an h2 heading to help your peer easily navigate within your code
- No imports are allowed, except for those explicitly mentioned in the section "Authorized functions" of the title block of each exercise
- You can use any built-in function as long as it is not prohibited in the exercise
- Save and load all the required data to the subfolder data/
- scikit-learn (0.23.1) is the library that you need for all the machine learning tasks.
- tqdm (4.46.1) is the library that you need for tracking progress

Chapter IV

Exercise 00 : Regularization

	Exercise 00
Regularization	
Turn-in directory : *ex*00/	
Files to turn in : `00_regularization.ipynb`	
Allowed functions : `no restrictions`	

In the previous day, you tried to solve different kinds of machine learning problems using different algorithms and gained an understanding of the basics. Today, you will try using more advanced techniques.

Let us start with regularization. Regularization in its broader sense is a technique that prevents a model from overfitting. As regards logistic regression, we have a formula with different Xs (features) and coefficients. Regularization penalizes coefficients that are too big making the formula more robust and more ready for the unknown data that it will have to work with in the future. L1 regularization makes some coefficients equal to zero. So this mode may be helpful for feature selection if there are lots of them and you need to reduce the number. L2 regularization does not make the weights equal to zero but may make them smaller. We cannot say: use only L2 regularization or L1. In the field of machine learning, there are not many things that are silver bullets. Usually, you need to try many different things on your dataset to find what suits it best.

If we talk about trees and forests, regularization is connected to the parameters that affect the number of cases in the leaves. If your tree is so thick that each leaf includes only one sample, the chances are that your tree has overfitted to the training dataset. To prevent that, you can play with such parameters as max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes, etc.

Many different algorithms have many different parameters of regularization. Our goal is not to cover them all. You should just know that they exist, and if you need them, you will understand how they work for a specific algorithm.

In this exercise, you will play with some of them. The dataset will be the same and the task is still to predict the weekday for each commit with data: uid, labname, numTrials, and hour of the commit.

What you need to do is fully described in the notebook:

Chapter V

Exercise 00 : Gridsearch

	Exercise 00
Gridsearch	
Turn-in directory : *ex*00/	
Files to turn in : `01_gridsearch.ipynb`	
Allowed functions : `no restrictions`	

We are sure that you got tired of iterating through the different parameters of various models manually. You probably think that there should be a way to automate it. Yes, there is GridSearch.

You can specify the range of values for the parameters that you want to optimize and put it to GridSearchCV. It will try all of them, calculate the metrics on cross-validation, and give you the best combination of parameters as well as the overall results of its mini-research. That is cool, right?

What you need to do describe in the notebook.

Chapter VI

Exercise 00 : Metrics

	Exercise 00
Metrics	
Turn-in directory : *ex*00/	
Files to turn in : `02_metrics.ipynb`	
Allowed functions : `no restrictions`	

Is 90% accuracy is a good result or not? Actually, it is not easy to say. Imagine a situation in which you have two classes that are unbalanced: 95% of the samples belong to the first class and 5% to the second. The accuracy will be worse than a naive classifier when we make predictions using the most popular class. And this is no fantasy. This case is quite popular for anti-fraud tasks, for example. The number of fraud cases is significantly lower than the number of normal cases. Is it a bad metric? What we are trying to say is that it is simple to understand and you can use it to compare different models within a task, but this metric can be misleading when you need to compare the results to a model from another task. Also, it does not say much about the errors. You just know how many of them there are. But of what kind?

There are some other metrics that can answer this question. They all come from the confusion matrix. The first is precision. It is the number of correctly predicted samples of one class divided by the number of predictions of that class. Imagine that we predicted 10 days as a weekend, but only 7 of them were really weekends. The precision is 0.7.

The second is recall. This is the number of correctly predicted samples of one class divided by the true number of that class. Imagine again that we predicted 10 days as a weekend but only 7 of them were really weekends and in the dataset there were 20 weekends. The recall is 0.35 (7/20).

Precision can be good when we want to show an ad that includes some 16+ content. We want to be precise in our prediction. Recall can be good for identifying terrorists. We may want to find them all no matter how many civilians experience some inconvenience along the way. There is also a metric that combines both of them in the harmonic mean F1 score. You can use it when you need to optimize both of them.

Also, there is the ROC-curve. When you make predictions, you usually you have probabilities. And the final classification is ade by comparing them to the threshold. For example, if we see that, for that given day, the probability of being a weekend is 0.2 and the threshold is 0.5, we can say that it is not a weekend. But if you change the threshold to 0.1, the same sample will get the prediction "weekend". Imagine now, that for each threshold we calculate recall and also the number of how many working days we predicted as weekends divided by the real number of working days. We can put both those values on a plot and we will get the ROC-curve. The higher it is, the better. Comparing curves can be rather inconvenient. That is why we can use another metric AUC (area under the curve). Precision, recall, and AUC are useful when we want to compare the performance of different models from different tasks. And they tell us something about the errors. Everything that we told you here was connected to binary classification. But it can be used with some adjustments for multiclass and multilabel classification. You can read about it here, for example.

What you need to do is fully described in the notebook.

Chapter VII

Exercise 00 : Ensembles

	Exercise 00
Ensembles	
Turn-in directory : *ex*00/	
Files to turn in : `03_ensembles.ipynb`	
Allowed functions : `no restrictions`	

You already know that a random forest is an ensemble of many different trees. But actually you can create an ensemble from any type of model. In this exercise, you will try several approaches: voting classifier, bagging classifier, and stacking classifier. Who knows, maybe it will help you increase the quality of your predictions.

What you need to do is fully described in the notebook.

Chapter VIII

Exercise 00 : Pipelines and OOP

	Exercise 00
Pipelines and OOP	
Turn-in directory : *ex*00/	
Files to turn in : `04_pipelines.ipynb`	
Allowed functions : `no restrictions`	

While trying to solve the problem you executed a lot of different actions: prepared the data, tried different models, tried different metrics, optimized their hyperparameters, and tried different kinds of ensembles. Now it probably looks a bit chaotic: your code is in different notebooks that require a lot of scrolling. In this exercise, the last of the day, you will make it look a bit cleaner, more organized. Why can this be important? In real life, you may want to share it with your colleagues, you may want to make it a part of your portfolio or it may be for your own future convenience. The chances are that if you get back to that code in several months, you will think: who made that mess?

In this exercise, you will try to apply the OOP approach to data analysis. The first part of your notebook will contain only the imports, classes and methods. The second part will be your "main program". You will work with the initial data and you will go through most of the steps that you executed before.

What you need to do is fully described in the notebook.