uday sai

4 Followers About

Is your Data Science Project Production ready?

uday sai May 24, 2020 · 5 min read

It is very common to write code in jupyter notebooks or spyder python files or pycharm files but it is not the intended way of deploying code in production environment. There are three ways of productionizing the ML code (I am not running into the details of using flask or flasgger or django framework and designing web application). These are just the ways for organizing the code and deploying the code in production environment. This gold standard of writable and readable code is prevalent in Developer's community and needs to be adapted by Data Scientists as well. Let's explore the three ways of writing the code for production environment:

northam%E2%80%99s-office-calls-emails-and-petition

> *Using routines and sub-routines*
>
> *Using classes and Custom Pipelines*
>
> *Using classes and Sklearn pipelines*

Using routines and Sub-routines

In this variant of code we design routines to accomplish the functionalities of different stages in data science project. For example, we will be designing routines for EDA, Feature Engineering, Outlier Detection and model deployment. We will use function calls to call routines and use the in appropriate block of code to accomplish end-to-end functionality of data science projects. However as the requirements increase the number of functions increases proportionally and it becomes quite difficult to manage the routines and sub-routines.

Using classes and Custom Pipelines

Before taking a deep dive into these pipelines let us explore some OOP concepts and concepts around Object-Oriented Programming. In object-oriented programming we write code in terms of "objects". The "objects" can store data, and can also store instructions or procedures to modify the data. The data is in the form of attributes and instructions or procedures are in the form of methods. In OOP the "objects" can learn and store this parameters. Parameters get automatically refreshed every time model is re-trained.

Fit and Transform : *The methods like fit and transform are used to learn parameters and to transform the data with learnt parameters.* To understand this let us consider the example of simpleImputer which has in built fit and transform methods which is used to learn mean, median or mode (based on our specification) and transform the dataset using the learnt mean, median or mode.

Pipeline : A pipeline is a set of data processing steps connected in series, where typically , the output of one element is the input of the next one. The elements of pipeline can be

A custom ML pipeline is therefore a sequence of steps, aimed at loading and transforming the data, to get it ready for training or scoring, where we write the processing steps as objects (OOP) and write the sequences as pipelines. We save one object, the pipeline, as a python pickle, with all the information needed to transform the raw inputs and get the predictions.

Advantages:

This can be tested, versioned, tracked and controlled and can develop future models on it. This is a good software developer practice to satisfy business needs.

Disadvantages:

This methodology requires a team of software developers to build and maintain. This causes a lot of overhead to Data Scientist for debugging or adding future models.

Using classes and Sklearn pipelines

Scikit-Learn is a python library that provides a solid implementation of a range of ML algorithms. Scikit-Learn provides efficient versions of a large number of common algorithms. It is characterized by a clean, uniform and streamlined API. Let us explore scikit-learn objects:

Transformers: The classes that have fit and transform method and transforms the data with a specific functionality. For example, the sklearn classes like Feature Selectors, Imputers etc… belong to this category.

Predictor/Estimator: A class that has fit and predict methods and predict the model which the data can fit. For example, the sklearn classes like LASSO, Decision Trees, SVM and Random Forests etc… belong to this category.

Pipeline: The classes that allows us to list and run transformers and predictors in sequence. All the steps should be transformers except the last one. The last one should be a predictor.

Inheritance is a popular OOPS concept in which we can acquire the methods and variables of the base class into new class which we build. We use the inbuilt get_params and set_params methods which are present in BaseEstimator. Besides we will use fit and transform methods present in TransformerMixin to design custom fit and transform methods.

Let us consider the below example, in this example we define a new class named NumericalTransformer which we inherit from BaseEstimator and TransformerMixin classes. In the code snippet you can see that __init__ method is used as constructor of class, fit method is used to learn parameters and as we have nothing to learn we just return self and in transform method we use the method of bottom-coding and if the height is less than or equal to 92 we convert all the values to 92.

```
from sklearn.base import BaseEstimator, TransformerMixin
class NumericalTransformer(BaseEstimator, TransformerMixin):
    #Class Constructor
    def __init__(self,variables=None ):
        if not isinstance(variables,list):
            self.variables=[variables]
        else:
            self.variables=variables
    #Return self, nothing else to do here
    def fit( self, X,y=None):
        return self
    #Custom transform method we wrote that creates aformentioned features and drops redundant ones
    def transform(self,X):
        #Check if needed
        X['height']=X['height'].apply(lambda x:92 if x<=92 else x)
        return X
```

Image: Custom Pipeline for Production

Conclusion:

Likewise we design custom classes and design a pipeline to execute these steps sequentially. Please refer the below code snippet to design a pipeline.

```
In [1]: import import_ipynb
        import config
        import preprocessors as pp
        from sklearn.pipeline import Pipeline
        from sklearn.impute import SimpleImputer
        from sklearn.tree import DecisionTreeClassifier

        importing Jupyter notebook from config.ipynb
        importing Jupyter notebook from preprocessors.ipynb

In [2]: full_pipe=Pipeline(
```

```
                    ('dec_tree',DecisionTreeClassifier(criterion='gini', max_depth= 5, min_samples_leaf=
1, min_samples_split= 2, splitter= 'best') )
                   ])
```

Image : Custom pipelines for Production

Eventually this pipeline is pickled (a serialization technique used in python to convert an object to stream of byte code and store in memory) and predictions on the test data can be made by this pickled pipeline. To illustrate this please look at the below code snippet:

Image: Pickling pipeline for production

References:

Please refer my github profile for full-code. Happy Learning…!!!

udaysai50/Real-World-Data-Science

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or...

github.com

Data Science Machine Learning Pipeline Deployment Transformers