Understanding Column Transformer and Machine Learning Pipelines

BEGINNER CLASSIFICATION MACHINE LEARNING PYTHON REGRESSION STRUCTURED DATA SUPERVISED

This article was published as a part of the Data Science Blogathon.

Overview

- **Understand Column Transformer**
- **Understand Machine learning Pipeline Structure**
- **Build End to End Machine Learning Pipeline**

Introduction

The dataset we get is not all the time clean and correct, It contains lots of inconsistency, missing values, and impurity which needs to handle and preprocess. Preprocessing the data cost lot of time and usually it is a messy task so to make the preprocessing steps easier and smooth Machine Learning contain a concept of Pipelining in which you can perform all the preprocessing steps step by step in a single chain.

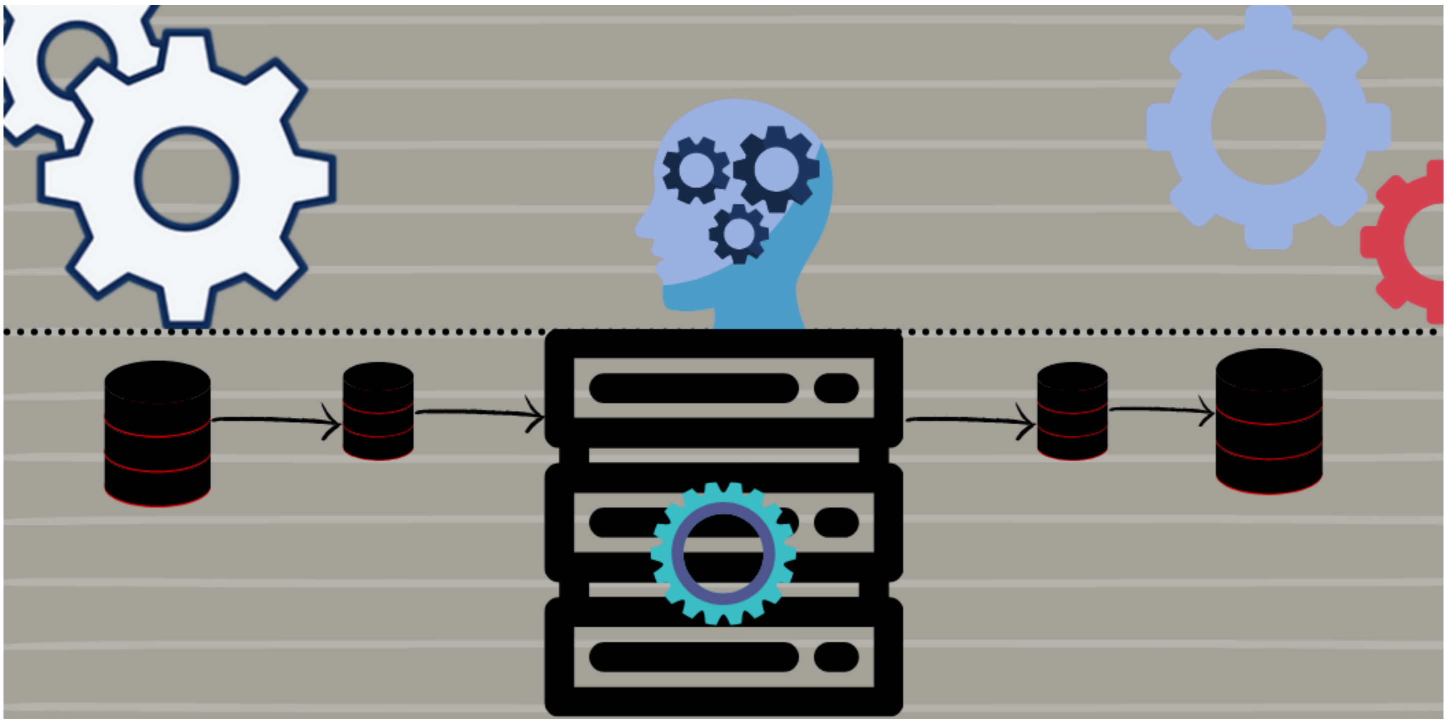


Table Of contents

Defining Problem

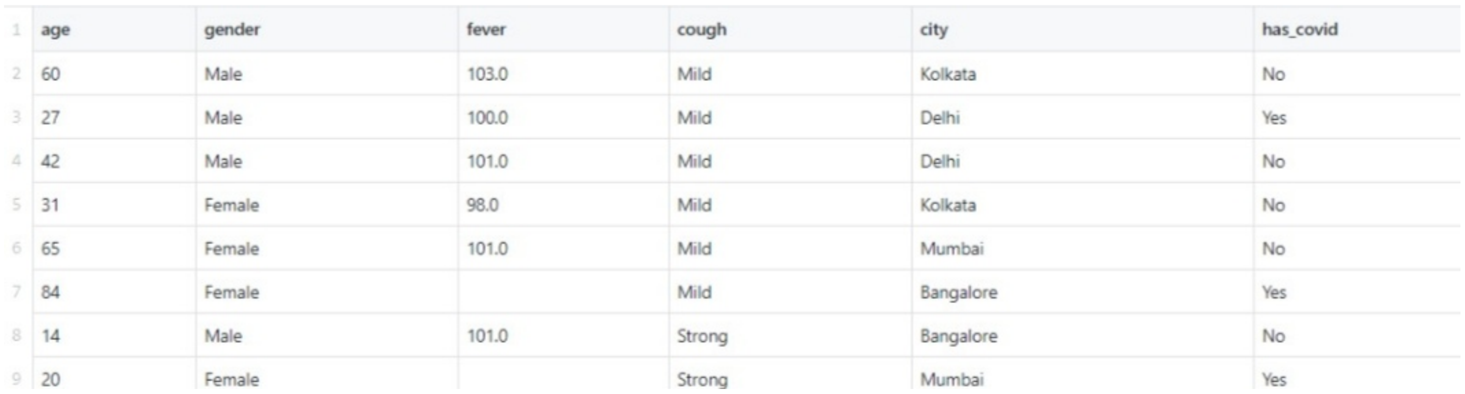
The problem with preprocessing data in separate steps looks good as a beginner or working for any competitions but when the aim is to build an end-end machine learning project for production purposes. Then before passing new data to model all, it's required to preprocess the data first and it's too hectic work to write all the preprocessing steps again so to save the hectic work time we make use of Pipelines.

While working with Machine learning pipelines, all preprocessing steps take place step by step in which Column Transformer helps us build it in a single line code.

Column Transformer

Column Transformer is a sciket-learn class used to create and apply separate transformers for numerical and categorical data. To create transformers we need to specify the transformer object and pass the list of transformations inside a tuple along with the column on which you want to apply the transformation.

To demonstrate how to build Column Transformation we are using a Covid Toy dataset which you get from Kaggle. this is how the dataset looks before any transformations



age	gender	fever	cough	city	has_covid
60	Male	103.0	Mild	Kolkata	No
27	Male	100.0	Mild	Delhi	Yes
42	Male	101.0	Mild	Delhi	No
31	Female	98.0	Mild	Kolkata	No
65	Female	101.0	Mild	Mumbai	No
84	Female		Mild	Bangalore	Yes
14	Male	101.0	Strong	Bangalore	No
20	Female		Strong	Mumbai	Yes

The transformations we will build is:

- Missing Value Imputation using Simple Imputer class
- Ordinal encoding using Ordinal Encoder
- Nominal encoding on countries using One Hot encoder

```
import pandas as pd from sklearn.impute import SimpleImputer from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder from sklearn.model_selection import train_test_split df =
pd.read_csv('covid_toy.csv')      X_train,X_test,y_train,y_test      =      train_test_split(df.drop(columns=
['has_covid']),df['has_covid'],test_size=0.2)    #create    Transformer    from    sklearn.compose    import
ColumnTransformer   transformer   =   ColumnTransformer(transformers=[   ('tnf1',SimpleImputer(),['fever']),
('tnf2',OrdinalEncoder(categories=[['Mild','Strong']]),['cough']),
('tnf3',OneHotEncoder(sparse=False,drop='first'),['gender','city'])              ],remainder='passthrough')
x_train_transform = transformer.fit_transform(X_train)
```

After applying the transformation you will get a Numpy array and the total columns transformed will be seven because the cities column with spread out into four different columns. And the resultant dataframe will look like this.

The remainder is initialized to passthrough means the column which we have not used in any transformation process should be pass as it is. If you want to drop the remaining column then initialize it with drop.

That is how easy Column Transformer makes the preprocessing step easy. I hope you have liked it.

Machine Learning Pipelines

Machine learning pipelines are a mechanism that chains multiple steps together so that the output of each step is used as input to the next step.

It means that it performs a sequence of steps in which the output of the first transformer becomes the input for the next transformer. If you have studied a little bit about neural networks then you can visualize the pipeline workflow that output of 1 layer feed as input for next layer.

Machine Learning Pipeline Design

Suppose for example you are working on a Machine learning project which has a too messy dataset containing missing values, categorical variables so you will first handle these problems and then train the model. But using Pipeline, you can chain all these 3 steps in a single step, making the project workflow smooth and easier.

Pipelines make it easy to apply the same preprocessing to train and test data. If you have previously deployed any machine learning model without using pipelines to any cloud server then you can experience how much complexities it creates to handle new data coming from a server.

Building Machine Learning Pipeline using Python

we are using a very popular dataset which you have previously used is a titanic dataset where you have to predict the survival rate of a passenger by using various features whether a person will survive or die.

you can download a dataset from here.

Planning a Machine Learning Pipeline

Before creating a pipeline just plan which transformations you need and you are going to implement in a Pipeline. In this example, we are going to create the Pipeline shown in the figure given below. Feature selection is not important But I wanted to show you, how you can use feature selection in a Pipeline.

Now start Building a Pipeline

1. Load a Dataset

```
import numpy as np import pandas as pd from sklearn.model_selection import train_test_split from sklearn.compose import ColumnTransformer from sklearn.impute import SimpleImputer from sklearn.preprocessing import OneHotEncoder from sklearn.preprocessing import MinMaxScaler from sklearn.pipeline import Pipeline,make_pipeline from sklearn.feature_selection import SelectKBest,chi2 from sklearn.tree import DecisionTreeClassifier df = pd.read_csv('titanic.csv')
```

we have loaded all the required libraries. we will use Decision Tree for training purpose

2. Train Test Split

```
#drop the non-required columns df.drop(columns=['PassengerId','Name','Ticket','Cabin'],inplace=True) X_train,X_test,y_train,y_test = train_test_split(df.drop(columns=['Survived']), df['Survived'], test_size=0.2, random_state=42)
```

Split the dataset into training and testing set to see how to apply the same pipeline on the train and test set.

3. Create Transformers

Create transformers which we are going to use in Pipeline. There are 4 transformers which we are using so let’s create them one by one. And one extra transformer is our Model so the total transformation steps are 5.

When working with Pipelines While creating a Column transformer it’s suggested to pass the index of columns rather than its name because after transformation it’s converted into Numpy Array and the array does not have any column names.

```
#1st Imputation Transformer trf1 = ColumnTransformer([ ('impute_age',SimpleImputer(),[2]), ('impute_embarked',SimpleImputer(strategy='most_frequent'),[6]) ],remainder='passthrough') #2nd One Hot Encoding trf2 = ColumnTransformer([ ('ohe_sex_embarked', OneHotEncoder(sparse=False, handle_unknown='ignore'),[1,6]) ], remainder='passthrough') #3rd Scaling trf3 = ColumnTransformer([ ('scale', MinMaxScaler(), slice(0,10)) ]) #4th Feature selection trf4 = SelectKBest(score_func=chi2,k=8) #5th Model trf5 = DecisionTreeClassifier()
```

In the first Imputation, Transformer applies the transform separately on two columns because different variable and different strategy to encode them is there. In the second Transformer of categorical Encoding, we can now apply both the columns together because now all things are happing in the same data, no new Numpy array is creating.

In the third Transformer, we have done Normalization on each column and the resultant dataframe will have 10 columns. In the fourth transformer, we select the top eight features that dominate the most for predicting the output. The last Transformer is our Machine Learning Model.

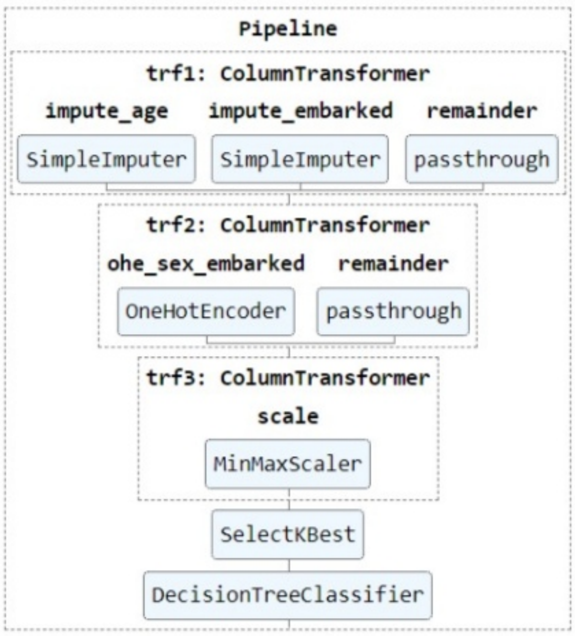
4) Create Pipeline

Implement a Pipeline using a sciket-learn class Pipeline and pass list of the tuple which will have two things first is the name of transformer and second is its object.

```
pipe = Pipeline([ ('trf1', trf1), ('trf2', trf2), ('trf3', trf3), ('trf4', trf4), ('trf5', trf5) ]) # Display
Pipeline from sklearn import set_config set_config(display='diagram') #fit data pipe.fit(X_train, y_train)
```

When we are using the Predictive model inside a pipeline use fit and predict function and whenever you are not using a model in the pipeline then you use the fit and transform function because at that time you are only preprocessing the data.

You can have a complete look-over to Pipeline how it works when you feed data to it.



5) Cross-Validation using Pipeline

Check the Performance of the algorithm using cross-validation with hyperparameter tuning. While initializing the hyperparameter you have to specify the name of the model object followed by the parameter.

```
from sklearn.model_selection import GridSearchCV from sklearn.model_selection import cross_val_score params =
{'trf5__max_depth':[1,2,3,4,5,None] } grid = GridSearchCV(pipe, params, cv=5, scoring='accuracy')
grid.fit(X_train, y_train) print(grid.best_score_)
```

6) Exporting the Pipeline

```
# export import pickle pickle.dump(pipe,open('pipe.pkl','wb'))
```

We have to use the pipeline in production so save the pipeline in a pickle format.

Difference between Pipeline and make_pipeline

The pipeline requires the naming of steps while make_pipeline does not and you can simply pass only the object of transformers. For the sake of simplicity, you should use make_pipeline.

SUMMARY

The pipeline is a mechanism that makes a data preprocessing task simple, easy, and time-saving. we have seen there are two types of pipelines created one is with algorithm and another without algorithm. Column Transformer is a way to make the data manipulation step easy and smooth. The pipeline gives your project a boost if you have developed a pipeline and used that in production then your project looks more attractive to the board.

The media shown in this article are not owned by Analytics Vidhya and is used at the Author's discretion.

Article Url - https://www.analyticsvidhya.com/blog/2021/05/understanding-column-transformer-and-machine-learning-pipelines/

agrawal@71