Easier Machine Learning with the New Column Transformer from Scikit-Learn

Rebecca Vickery
Oct 1, 2018 · 5 min read

Photo by Samule Sun on Unsplash

Last week scikit-learn released version 0.20.0, one of the features in this release I am most excited about is the ColumnTransformer. This function allows you to combine several feature extraction or transformation methods into a single transformer. Say, you are working on a machine learning problem, and you have a dataset containing a mixture of categorical and numerical columns. Rather than having to handle each of

these separately, and perhaps writing a function to then apply this to new data. These can now be combined into a transformer which can easily be reapplied, and extended.

This is a part of my machine learning workflow I have been keen to simplify, so over the weekend I took a simple data set I had lying around on my laptop, and had a go at applying this to a classification problem.

The data set I am using was taken from the Analytics Vidhya loan prediction competition and can be downloaded here. The purpose of this competition is to predict wether or not a loan application will be successful based on a number of customer features. This contains both categorical and numerical variables, and is a nice simple data set to practice using the new ColumnTransformer.

Data Preparation

To start with I am using pandas to read in both files. Using the dtypes function I can see that there are both numerical and categorical features present in the dataset.

```
import pandas as pd
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
print(train.shape, test.shape)
print(train.dtypes)
```

```
((614, 13), (367, 12))
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

Before going further I am going to drop the Loan_ID column as that will not be used in the model. I am also filling any null values with the most commonly occurring value for each column. There are of course a number of methods I could choose for this but as I am just trying out a new function I am not too worried about the accuracy of the model for now.

```
train = train.drop(['Loan_ID'], axis=1)
test =  test.drop(['Loan_ID'], axis=1)

train = train.apply(lambda x:x.fillna(x.value_counts().index[0]))
test = test.apply(lambda x:x.fillna(x.value_counts().index[0]))
```

Next I'm going to specify the features (X) and target (y). I then use the sklearn train_test_split function to divide the training data into test and train ready to train and validate the model.

```
feature_set = train.drop(['Loan_Status'], axis=1)
X = feature_set.columns[:len(feature_set.columns)]
y = 'Loan_Status'

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    train[X], train[y], random_state=0)
```

ColumnTransformer

The next step is to apply transformation to columns to optimise them for use in the classification model. Usually I would write a function that applies these transformations, so that I can re-use it on both the test and train set, and any holdout data I may have for later validation. However, I am going to try using the ColumnTransformer to simplify these steps.

For simplicity, again I am not too worried about the accuracy of this model, I am going to transform the categorical columns using the sklearn OneHotEncoder. I will also normalize the numerical columns using the Normalizer function.

The ColumnTransformer takes a list of tuples specifying the transformers, and the corresponding columns on which the transformation needs to be applied. The columns can either be entered as strings specifying the column names in a pandas data frame, or as in the code I have used below, as integers which are interpreted as the column positions.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import Normalizer, OneHotEncoder
```

```
colT = ColumnTransformer(
    [("dummy_col", OneHotEncoder(categories=[['Male', 'Female'],
                                             ['Yes', 'No'],
                                              ['0','1', '2','3+'],
                                              ['Graduate', 'Not
Graduate'],
                                              ['No', 'Yes'],
                                              ['Semiurban', 'Urban',
'Rural']]), [0,1,2,3,4,10]),
      ("norm", Normalizer(norm='l1'), [5,6,7,8,9])])
```

You will notice in the code above I have used the categories argument of the OnHotEncoder function. This takes a list of all possible categories in each column as a list of lists. This produces one hot encoded columns for all categories even if data does not exist for that category in the column. The reason for doing this is that when using the ColumnTransformer function on new data. If it doesn't contain the same categories in each feature then the array produced will not be the same shape as the data used to train the model, and you will get an error.

I apply the transformer to the training data as shown below. The first few rows of the output, which is a list of lists containing numerical arrays, is shown beneath the code.

```
X_train = colT.fit_transform(X_train)
X_train
```

```
array([[1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 9.23921822e-01,
        0.00000000e+00, 1.77677273e-02, 5.81489259e-02, 1.61524794e-04],
       [1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 6.07544934e-01,
        2.96267035e-01, 2.48864310e-02, 7.11040885e-02, 1.97511357e-04],
       [1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        1.00000000e+00, 0.00000000e+00, 1.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 6.78518116e-01,
        2.44266522e-01, 2.82263536e-02, 4.88533044e-02, 1.35703623e-04],
```

To transform the X_test data you simply apply the column transformer again.

```
X_test = colT.transform(X_test)
```

Training the Model

The data is now ready to be used in any scikit-learn classifier. For simplicity I have just used a RandomForestClassifier model with the default parameters.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)
y_pred = random_forest.predict(X_test)
print(classification_report(y_test, y_pred, target_names=['Y',
'N']))
```

	precision	recall	f1-score	support
Y	0.66	0.58	0.62	43
N	0.84	0.88	0.86	111
micro avg	0.80	0.80	0.80	154
macro avg	0.75	0.73	0.74	154
weighted avg	0.79	0.80	0.79	154

Predicting New Data

To test how the ColumnTransformer would work if we were to use this model to make predictions on previously unseen data. I took a sample of rows from the test csv file I read in earlier. I then simply re-use the column transformer to apply the preprocessing steps.

```
test_samp = test[:15]
test_samp = colT.transform(test_samp)

random_forest.predict(test_samp)
```

Applying the Random Forest model to predict the loan status gives the following output.

```
array(['Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y',
       'N', 'Y'], dtype=object)
```

Having used this new tool I have to say I am impressed. It has really simplified my workflow and I am going to be using this a lot in my work from now on. It is really simple to add new transformation steps into the ColumnTransformer, something which I will definitely be looking at over the next few weeks.

Machine Learning Scikit Learn Data Science Data Preprocess Python