**Creating Custom Transformer**

In the last section, you used a custom transformer that extracted whether each text started with a verb. You can implement a custom transformer yourself by extending the base class in Scikit-Learn. Let's take a look at a a very simple example that multiplies the input data by ten.

**import** numpy **as** np

**from** sklearn.base **import** BaseEstimator, TransformerMixin

**class** **TenMultiplier**(BaseEstimator, TransformerMixin):

**def** **\_\_init\_\_**(self):

**pass**

**def** **fit**(self, X, y=None):

**return** self

**def** **transform**(self, X):

**return** X \* 10

Remember, all estimators have a fit method, and since this is a transformer, it also has a transform method.

* **FIT METHOD:** This takes in a 2d array X for the feature data and a 1d array y for the target labels. Inside the fit method, we simply return self. This allows us to chain methods together, since the result on calling fit on the transformer is still the transformer object. This method is required to be compatible with scikit-learn.
* **TRANSFORM METHOD:** The transform function is where we include the code that well, transforms the data. In this case, we return the data in X multiplied by 10. This transformmethod also takes a 2d array X.

Let's test our new transformer, by entering the code below in the interactive python interpreter in the terminal, ipython. We can also do this in Jupyter notebook.

multiplier = TenMultiplier()

X = np.array([6, 3, 7, 4, 7])

multiplier.transform(X)

This outputs the following:

array([60, 30, 70, 40, 70])

Nice! Next, we'll create a custom transformer that has a bit more significance. Let's build a case normalizer, which simply converts all text to lowercase. We aren't setting anything in our init method, so we can actually remove that. We can leave our fit method as is, and focus on the transform method. We can lowercase all the values in X by applying a lambda function that calls lower on each value. We'll have to wrap this in a pandas Series to be able to use this apply function.

**import** numpy **as** np

**import** pandas **as** pd

**from** sklearn.base **import** BaseEstimator, TransformerMixin

**class** **CaseNormalizer**(BaseEstimator, TransformerMixin):

**def** **fit**(self, X, y=None):

**return** self

**def** **transform**(self, X):

**return** pd.Series(X).apply(**lambda** x: x.lower()).values

case\_normalizer = CaseNormalizer()

X = np.array(['Implementing', 'a', 'Custom', 'Transformer', 'from', 'SCIKIT-LEARN'])

case\_normalizer.transform(X)

Entering the code above in ipython outputs the following:

array(['implementing', 'a', 'custom', 'transformer', 'from',

'scikit-learn'], dtype=object)

Awesome! It's a good idea to learn how to write your own custom functions - it allows you to have more control and flexibility with your machine learning pipelines.

Another way to create custom transformers is by using this FunctionTransformer from scikit-learn's preprocessing module. This allows you to wrap an existing function to become a transformer. This provides less flexibility, but is much simpler. You can learn more about this in the link below.

Read more about using FunctionTransformer to create custom transformers [**here**](http://scikit-learn.org/stable/modules/preprocessing.html#custom-transformers) and [**here**](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.FunctionTransformer.html#sklearn.preprocessing.FunctionTransformer).