Custom Transformers

Scikit-Learn provides many useful transformers, some times we have to write our own for task such as custom cleanup operations or combining specific attributes. These custom transformers work seamlessly with Scikit-Learn functionalities such as 'pipeline' .

Before we move on to building custom transformers we have to know a little bit about Scikit-Learn.

Scikit-Learn API Consistency: All objects share a consistent and simple interface.

Estimators – Any objects that can estimate some parameters based on a data-set is call estimator (e.g. sklearn.linear_model.LinearRegression). The estimation is performed by **fit()** method, **fit()** method takes data-set as parameter (two data-sets for supervised learning, the second parameter contains labels). Any other parameter needed to guide the estimation process is considered **hyperparameter** (n_jobs – is a one example for number processors for LinearRegression). The **hyperparameter** must be set as an instance variable (generally via a constructor parameter).

Transformers – Any object that transform data are called as transformer, some estimators (such as an imputer) can also transform a data-set. The transformation is performed by the **transform()** method, the data-set is the parameter for the method. The method returns transformed data-set. The transformation generally relies on the learned parameters (output from fit() method). All transformers have a convenient method **fit_transform()**. This is equivalent to calling fit() and then **transform()**. Some times **fit_transform()** method is optimized and runs much faster.

Predictors – Some estimators are capable of making prediction given a data-set (example LinearRegression). The **predict()** method takes data-set and returns a data-set of corresponding predictions. It also has **score()** method that measures the quality of the predictions given a test set (corresponding labels in case of supervised learning).

Inspection – All the estimator's hyper-parameters are accessible directly via public instance variables (e.g., imputer.strategy), and all the estimator's learned parameters are also accessible via public instance variables with an underscore suffix (e.g., imputer.statistics_)

Nonproliferation of classes – Data-sets are represented as NumPy arrays or SciPy sparse matrices, instead of homemade classes.
Hyperparameters are just regular Python strings or numbers.

Composition – Existing building blocks are reused as much as possible. Example, it is easy to create a **Pipeline** estimator from an arbitrary sequence of transformers followed by a final esimator, as we will see.

Sensible defaults – Scikit-Learn provides reasonable default values for most parameters, making it easy to create a baseline working system quickly.

Pipeline – A sequence of data processing components is called a data pipeline. Pipelines are very common in Machine Learning systems, since there is a lot of data to manipulate and many transformations to apply.