**Pipelines And Feature Unions**

* **FEATURE UNION:** [**Feature union**](http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html) is a class in scikit-learn’s Pipeline module that allows us to perform steps in parallel and take the union of their results for the next step.
* A *pipeline* performs a list of steps in a *linear sequence*, while a *feature union* performs a list of steps in *parallel* and then combines their results.
* In more complex workflows, multiple feature unions are often used within pipelines, and multiple pipelines are used within feature unions.

Sometimes, you don't always have all the data transformation steps you need in scikit-learn's library, which is why it is possible to actually create your own custom transformers. For the video below, just keep in mind that TextLengthExtractor is a custom transformer that is already built in a separate file and imported for this example.

**Using Feature Union**

Taking the example from the previous video, let's say you wanted to extract two different kinds of features from the same text column - tfidf values, and the length of the text. Your first approach might be to create an additional column from the text column called text\_length like this. Then both textand text\_length can be part of your feature matrix. But now your pipeline would break. You can't run CountVectorizer on NumPy arrays of strings and integers.

df['txt\_length'] = df['text'].apply(len)

X = df[['text', 'txt\_length']].values

y = df['label'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

pipeline = Pipeline([

('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('clf', RandomForestClassifier()),

])

*# train classifier*

pipeline.fit(Xtrain)

*# predict on test data*

predicted = pipeline.predict(Xtest)

Let's say you had a custom transformer called TextLengthExtractor. Now, you could leave X\_train as just the original text column, if you could figure out how to add the text length extractor to your pipeline. If only you could fit it on the original text data, rather than the output of the previous transformer. But you need both the outputs of TfidfTransformer and TextLengthExtractor to feed into the classifier as input.

X = df['text'].values

y = df['label'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

pipeline = Pipeline([

('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('txt\_length', TextLengthExtractor()),

('clf', RandomForestClassifier()),

])

*# train classifier*

pipeline.fit(Xtrain)

*# predict on test data*

predicted = pipeline.predict(Xtest)

* Feature unions are super helpful for handling these situations, where we need to run two steps in parallel on the same data and combine their results to pass into the next step.
* Like pipelines, feature unions are built using a list of (key, value) pairs, where the key is the string that you want to name a step, and the value is the estimator object. Also like pipelines, feature unions combine a list of estimators to become a single estimator. However, a feature union runs its estimators in parallel, rather than in a sequence as a pipeline does. In this example, the estimators run in parallel are nlp\_pipeline and text\_length. Notice we use a pipeline in this feature union to make sure the count vectorizer and tfidf transformer steps are still running in sequence.

X = df['text'].values

y = df['label'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

pipeline = Pipeline([

('features', FeatureUnion([

('nlp\_pipeline', Pipeline([

('vect', CountVectorizer()

('tfidf', TfidfTransformer())

])),

('txt\_len', TextLengthExtractor())

])),

('clf', RandomForestClassifier())

])

*# train classifier*

pipeline.fit(Xtrain)

*# predict on test data*

predicted = pipeline.predict(Xtest)

* Now, our pipeline doesn't break and uses both features! This would be equivalent to this code.

vect = CountVectorizer()

tfidf = TfidfTransformer()

txt\_len = TextLengthExtractor()

clf = RandomForestClassifier()

*# train classifier*

X\_train\_counts = vect.fit\_transform(X\_train)

X\_train\_tfidf = tfidf.fit\_transform(X\_train\_counts)

X\_train\_len = txt\_len.fit\_transform(X\_train)

X\_train\_features = hstack([X\_train\_tfidf, X\_train\_len])

clf.fit(X\_train\_features, y\_train)

*# predict on test data*

X\_test\_counts = vect.transform(X\_test)

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

X\_test\_len = txt\_len.transform(X\_test)

X\_test\_features = hstack([X\_test\_tfidf, X\_test\_len])

y\_pred = clf.predict(X\_test\_features)

* The tfidf transformer and the text length extractor are fit to the input data, in this case the raw data, independently. They are then performed in parallel, and their outputs are combined and passed to the next estimator, in this case, the classifier.

Read more about feature unions in Scikit-learn's [**user guide**](http://scikit-learn.org/stable/modules/pipeline.html#feature-union).