Pipelines, feature & text preprocessing

The pipeline workflow

- Repeatable way to go from raw data to trained model
- Pipeline object takes sequential list of steps
    - Output of one step is input to next step
- Each step is a tuple with two elements
    - Name: string
    - Transform: obj implementing .fit() and .transform()
- Flexible: a step can itself be another pipeline!

Instantiate simple pipeline with one step

```
In [1]: from sklearn.pipeline import Pipeline

In [2]: from sklearn.linear_model import LogisticRegression

In [3]: from sklearn.multiclass import OneVsRestClassifier

In [4]: pl = Pipeline([
   ...:         ('clf', OneVsRestClassifier(LogisticRegression()))
   ...:     ])
```

Train and test with sample numeric data

```
In [5]: sample_df.head()
Out[5]:
  label    numeric     text  with_missing
0     a  -4.167578      bar     -4.084883
1     a  -0.562668               2.043464
2     a -21.361961             -33.315334
3     b  16.402708  foo bar     30.884604
4     a -17.934356      foo    -27.488405
```

Train and test with sample numeric data

```
In [6]: from sklearn.model_selection import train_test_split

In [7]: X_train, X_test, y_train, y_test = train_test_split(
   ...:                                         sample_df[['numeric']],
   ...:                                         pd.get_dummies(sample_df['label']),
   ...:                                         random_state=2)

In [8]: pl.fit(X_train, y_train)
Out[8]:
Pipeline(steps=[('clf', OneVsRestClassifier(estimator=LogisticRegression(C=1.0,
class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
          n_jobs=1))])
```

Train and test with sample numeric data

```
In [9]: accuracy = pl.score(X_test, y_test)

In [10]: print('accuracy on numeric data, no nans: ', accuracy)
accuracy on numeric data, no nans:  0.44
```

Adding more steps to the pipeline

```
In [11]: X_train, X_test, y_train, y_test = train_test_split(sample_df[['numeric',
    ...:                                        'with_missing']], pd.get_dummies(
    ...:                                        sample_df['label']), random_state=2)

In [12]: pl.fit(X_train, y_train)
Traceback (most recent call last):
  ...
ValueError: Input contains NaN, infinity or a value too large for
dtype('float64').
```

Preprocessing numeric features with missing data

```
In [13]: from sklearn.preprocessing import Imputer

In [14]: X_train, X_test, y_train, y_test = train_test_split(sample_df[['numeric',
    ...:                                                    'with_missing']], pd.get_dummies(
    ...:                                                    sample_df['label']), random_state=2)

In [15]: pl = Pipeline([
   ....:             ('imp', Imputer()),
   ....:             ('clf', OneVsRestClassifier(LogisticRegression()))
   ....:     ])
```

Preprocessing numeric features with missing data

```
In [16]: pipeline.fit(X_train, y_train)

In [17]: accuracy = pl.score(X_test, y_test)

In [18]: print('accuracy on all numeric, incl nans: ', accuracy)
accuracy on all numeric, incl nans:  0.48
```

- No errors!

Let’s practice!

Text features and feature unions

Preprocessing text features

```
In [1]: from sklearn.feature_extraction.text import CountVectorizer

In [2]: X_train, X_test, y_train, y_test = train_test_split(sample_df['text'],
   ...:                                                     pd.get_dummies(
   ...:                                                     sample_df['label']),
   ...:                                                     random_state=2)

In [3]: pl = Pipeline([
   ....:         ('vec', CountVectorizer()),
   ....:         ('clf', OneVsRestClassifier(LogisticRegression()))
   ....:     ])
```

Preprocessing text features

```
In [4]: pl.fit(X_train, y_train)
Out[4]:
Pipeline(steps=[('vec', CountVectorizer(analyzer='word', binary=False,
decode_error='strict', dtype=<class 'numpy.int64'>, encoding='utf-8',
input='content', lowercase=True, max_df=1.0, max_features=None, min_df=1,
ngram_range=(1, 1), preprocessor=None, stop_words=None, strip_...=None,
solver='liblinear', tol=0.0001, verbose=0, warm_start=False), n_jobs=1))])

In [5]: accuracy = pl.score(X_test, y_test)

In [6]: print('accuracy on sample data: ', accuracy)
accuracy on sample data:  0.64
```

Preprocessing multiple dtypes

- Want to use <u>all</u> available features in one pipeline
- Problem
  - Pipeline steps for numeric and text preprocessing can't follow each other
  - e.g., output of CountVectorizer can't be input to Imputer
- Solution
  - FunctionTransformer() & FeatureUnion()

FunctionTransformer

- Turns a Python function into an object that a scikit-learn pipeline can understand
- Need to write two functions for pipeline preprocessing
    - Take entire DataFrame, return numeric columns
    - Take entire DataFrame, return text columns
- Can then preprocess numeric and text data in separate pipelines

Putting it all together

```
In [7]: X_train, X_test, y_train, y_test = train_test_split(sample_df[['numeric',
   ...:                                          'with_missing', 'text']], pd.get_dummies(
   ...:                                          sample_df['label']), random_state=2)

In [8]: from sklearn.preprocessing import FunctionTransformer

In [9]: from sklearn.pipeline import FeatureUnion
```

Putting it all together

```
In [10]: get_text_data = FunctionTransformer(lambda x: x['text'],
    ...:                                     validate=False)

In [11]: get_numeric_data = FunctionTransformer(lambda x: x[['numeric',
    ...:                                     'with_missing']], validate=False)
```

FeatureUnion Text and Numeric Features

```
In [12]: from sklearn.pipeline import FeatureUnion

In [13]: union = FeatureUnion([
    ...:         ('numeric', numeric_pipeline),
    ...:         ('text', text_pipeline)
    ...:     ])
```

Text Features

Numeric Features

Putting it all together

```
In [14]: numeric_pipeline = Pipeline([
    ...:                         ('selector', get_numeric_data),
    ...:                         ('imputer', Imputer())
    ...:                     ])

In [15]: text_pipeline = Pipeline([
     ...:                        ('selector', get_text_data),
     ...:                        ('vectorizer', CountVectorizer())
     ...:                    ])

In [16]: pl = Pipeline([
     ...:     ('union', FeatureUnion([
     ...:         ('numeric', numeric_pipeline),
     ...:         ('text', text_pipeline)
     ...:     ])),
     ...:     ('clf', OneVsRestClassifier(LogisticRegression()))
     ...:  ])
```

Let's practice!

MACHINE LEARNING WITH THE EXPERTS

Choosing a classification model

Main dataset: lots of text

```
In [1]: LABELS = ['Function', 'Use', 'Sharing', 'Reporting', 'Student_Type',
   ...:           'Position_Type', 'Object_Type',  'Pre_K', 'Operating_Status']

In [2]: NON_LABELS = [c for c in df.columns if c not in LABELS]

In [3]: len(NON_LABELS) - len(NUMERIC_COLUMNS)
Out[3]: 14
```

Using pipeline with the main dataset

```
In [4]: import numpy as np

In [5]: import pandas as pd

In [6]: df = pd.read_csv('TrainingSetSample.csv', index_col=0)

In [7]: dummy_labels = pd.get_dummies(df[LABELS])

In [8]: X_train, X_test, y_train, y_test = multilabel_train_test_split(
   ...:                                    df[NON_LABELS], dummy_labels,
   ...:                                    0.2)
```

Using pipeline with the main dataset

```
In [10]: get_text_data = FunctionTransformer(combine_text_columns,
    ...:                                     validate=False)

In [11]: get_numeric_data = FunctionTransformer(lambda x:
    ...:                      x[NUMERIC_COLUMNS], validate=False)

In [12]: pl = Pipeline([
    ...:         ('union', FeatureUnion([
    ...:                 ('numeric_features', Pipeline([
    ...:                     ('selector', get_numeric_data),
    ...:                     ('imputer', Imputer())
    ...:                 ])),
    ...:                 ('text_features', Pipeline([
    ...:                     ('selector', get_text_data),
    ...:                     ('vectorizer', CountVectorizer())
    ...:                 ]))
    ...:              ])
    ...:         ),
    ...:         ('clf', OneVsRestClassifier(LogisticRegression()))
    ...:     ])
```

Performance using main dataset

```
In [13]: pl.fit(X_train, y_train)
Out[13]:
Pipeline(steps=[('union', FeatureUnion(n_jobs=1,
transformer_list=[('numeric_features', Pipeline(steps=[('selector',
FunctionTransformer(accept_sparse=False, func=<function <lambda> at
0x11415ec80>, pass_y=False, validate=False)), ('imputer', Imputer(axis=0,
copy=True, missing_valu...=None, solver='liblinear', tol=0.0001, verbose=0,
warm_start=False),n_jobs=1))])
```

Flexibility of model step

- Is current model the best?
- Can quickly try different models with pipelines
    - Pipeline preprocessing steps unchanged
    - Edit the model step in your pipeline
    - Random Forest, Naïve Bayes, k-NN

Easily try new models using pipeline

```
In [14]: from sklearn.ensemble import RandomForestClassifier

In [15]: pl = Pipeline([
    ...:         ('union', FeatureUnion(
    ...:             transformer_list = [
    ...:                 ('numeric_features', Pipeline([
    ...:                     ('selector', get_numeric_data),
    ...:                     ('imputer', Imputer())
    ...:                 ])),
    ...:                 ('text_features', Pipeline([
    ...:                     ('selector', get_text_data),
    ...:                     ('vectorizer', CountVectorizer())
    ...:                 ]))
    ...:              ]
    ...:         )),
    ...:         ('clf', OneVsRest(RandomForestClassifier()))
    ...:     ])
```

Let's practice!