Pandas, Pipelines, and Custom Transformers

Julie Michelman, Data Scientist, zulily

PyData Seattle 2017

July 6, 2017

Outline

- Background
  - Refresher on pandas DataFrames, sklearn modeling
  - Introduce dataset
- Transformers
  - Built-in and custom transformers
  - Combining transformers with Pipelines
- pandas + Pipelines
  - Challenge and solution
  - Fit a better model

Background

`pandas DataFrame`

- Two-dimensional, tabular data structure
- Columns can be different types
  - More flexible than `numpy arrays`
- Rows labeled with index
  - Defaults to row numbers
  - Multi-index possible

Dataset

- Special Events Permits
  - City of Seattle Open Data portal
  - “Applications received and Special Event permits issued by Special Events Office, Office of Economic Development, since 2014”
- Minor preprocessing
  - Filtered to 2016
  - Column names to lower_case_with_underscores

Data source: https://data.seattle.gov/Permitting/Special-Events-Permits/dm95-f8w5

Dataset – Let's Take a Look

```
>>> import pandas as pd
>>> df = pd.read_csv('Special_Events_Permits_2016.csv')
>>> df.head()
```

```
         application_date permit_status    permit_type event_category  \
0  12/09/2015 12:00:00 AM      Complete  Special Event       Athletic
1  01/07/2016 12:00:00 AM      Complete  Special Event     Commercial
2  01/21/2016 12:00:00 AM      Complete  Special Event      Community
3  01/21/2016 12:00:00 AM      Complete  Special Event      Community
4  01/21/2016 12:00:00 AM      Complete  Special Event      Community
```

More columns

```
  event_sub_category                                name_of_event  \
0          Run/Walk  See Jane Run Women's Half Marathon and 5K
1                NaN                    Capitol Hill Block Party
2                NaN      Sounders FC March to the Match (7.09)
3                NaN      Sounders FC March to the Match (7.13)
4                NaN      Sounders FC March to the Match (7.31)

  year_month_app        event_start_date          event_end_date  \
0      S16JY044  07/10/2016 12:00:00 AM  07/10/2016 12:00:00 AM
1      S16JY046  07/22/2016 12:00:00 AM  07/24/2016 12:00:00 AM
2      S16JY074  07/09/2016 12:00:00 AM  07/09/2016 12:00:00 AM
3      S16JY075  07/13/2016 12:00:00 AM  07/13/2016 12:00:00 AM
4      S16JY076  07/31/2016 12:00:00 AM  07/31/2016 12:00:00 AM
```

And more columns

```
 event_location_park event_location_neighborhood council_district precinct  \
0      Gas Works Park      Multiple Neighborhoods                3    North
1                 NaN                Capitol Hill                3     East
2                 NaN             Pioneer Square                 3     West
3                 NaN             Pioneer Square                 3     West
4                 NaN             Pioneer Square                 3     West

                organization  attendance
0                 See Jane Run      4500.0
1  Independent Event Solutions     27000.0
2          Seattle Sounders FC       705.0
3          Seattle Sounders FC       705.0
4          Seattle Sounders FC       705.0
```

Build a Machine Learning Model

- Outcome – `permit_status`
  - Binary: Will event be “Complete” or not?

- Features
  - Everything else!
  - Raw, transformed, combinations, etc.

Modeling with `scikit-learn` – Set up

- Set aside test data – No peeking!

```
>>> from sklearn.model_selection import train_test_split
>>> df_train, df_test = train_test_split(df)
```

- Define outcome, one feature

```
>>> import numpy as np
>>> y_train = np.where(df_train.permit_status ==
...     'Complete', 1, 0)
>>> X_train = df_train[['attendance']].fillna(value=0)
```

Modeling with `scikit-learn` – Fit Model

- Create a model object

```
>>> from sklearn.linear_model import LogisticRegression
>>> model = LogisticRegression()
```

- Fit model and predict on training data

```
>>> model.fit(X_train, y_train)
>>> y_pred_train = model.predict(X_train)
>>> p_pred_train = model.predict_proba(X_train)[:, 1]
```

Modeling with `scikit-learn` – Evaluation

- Predict on test data

```
>>> p_baseline = [y_train.mean()]*len(y_test)
>>> p_pred_test = model.predict_proba(X_test)[:, 1]
```

- Measure performance on test data

```
>>> from sklearn.metrics import roc_auc_score
>>> auc_base = roc_auc_score(y_test, p_baseline)  # 0.50
>>> auc_test = roc_auc_score(y_test, p_pred_test) # 0.58
```

Transformers

`scikit-learn` Transformers & Estimators

Transformer

- For data preparation
- `fit` – find parameters from training data (if needed)
- `transform` – apply to training or test data

Estimator

- For modeling
- `fit` – find parameters from training data
- `predict` – apply to training or test data

`scikit-learn` Transformers & Estimators

Transformer

- `StandardScaler`
- `fit` – find mean, standard deviation of each feature
- `transform` – subtract mean, then divide by sd

Estimator

- `LogisticRegression`
- `fit` – find coefficients in logistic regression formula
- `predict` – plug into formula, get predicted class

Multiple Transformers

- Go through several transformation steps

```
>>> from sklearn.preprocessing import (Imputer,
...     PolynomialFeatures, StandardScaler)
>>> imputer = Imputer()
>>> quadratic = PolynomialFeatures()
>>> standardizer = StandardScaler()
```

Multiple Transformers

- No one wants to write this:

```
>>> X_train_imp = imputer.fit_transform(X_train_raw)
>>> X_train_quad = quadratic.fit_transform(X_train_imp)
>>> X_train = standardizer.fit_transform(X_train_quad)
```

- And then this:

```
>>> X_test_imp = imputer.transform(X_test_raw)
>>> X_test_quad = quadratic.transform(X_test_imp)
>>> X_test = standardizer.transform(X_test_quad)
```

`Pipelines` to the Rescue!

- Instead, put steps together in a `Pipeline`

```
>>> from sklearn.pipeline import Pipeline
>>> pipeline = Pipeline([
...     ('imputer', Imputer()),
...     ('quadratic', PolynomialFeatures()),
...     ('standardizer', StandardScaler())
... ])
```

Awesome blog post: http://zacstewart.com/2014/08/05/pipelines-of-featureunions-of-pipelines.html

`Pipelines` to the Rescue!

- And just write this:

```
>>> X_train = pipeline.fit_transform(X_train_raw)
>>> X_test = pipeline.transform(X_test_raw)
```

Transformers in Parallel

- Try more than one transformation
- Put steps together with a `FeatureUnion`

```
>>> from sklearn.pipeline import FeatureUnion
>>> feature_union = FeatureUnion([
...     ('fill_avg', Imputer(strategy='mean')),
...     ('fill_mid', Imputer(strategy='median')),
...     ('fill_freq', Imputer(strategy='most_frequent'))
... ])
```

Transformers in Parallel

- And just write this:

```
>>> X_train = feature_union.fit_transform(X_train_raw)
>>> X_test = feature_union.transform(X_test_raw)
```

EDA – event_location_park

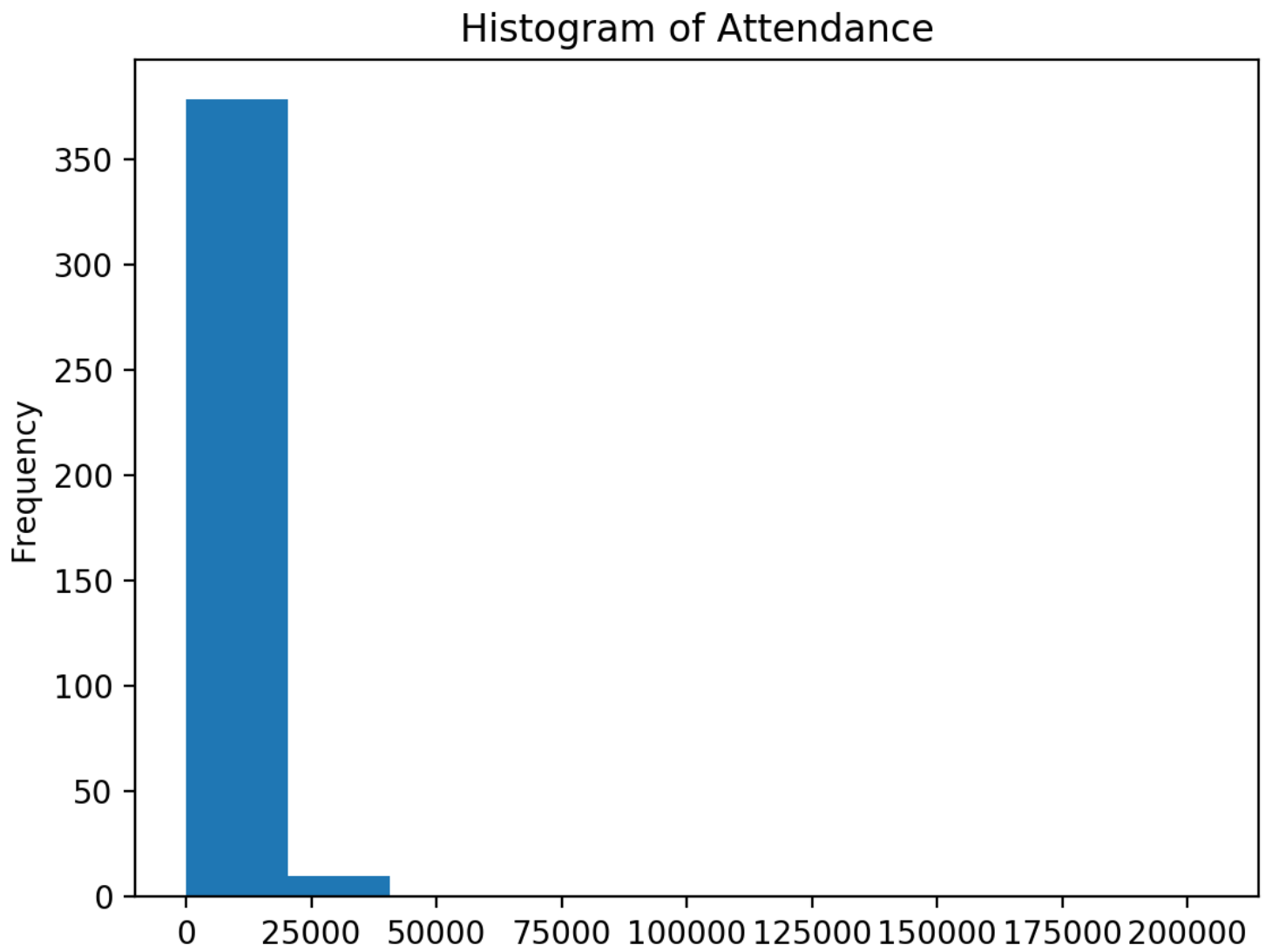
- Event Park: mostly N/A, not exhaustive list

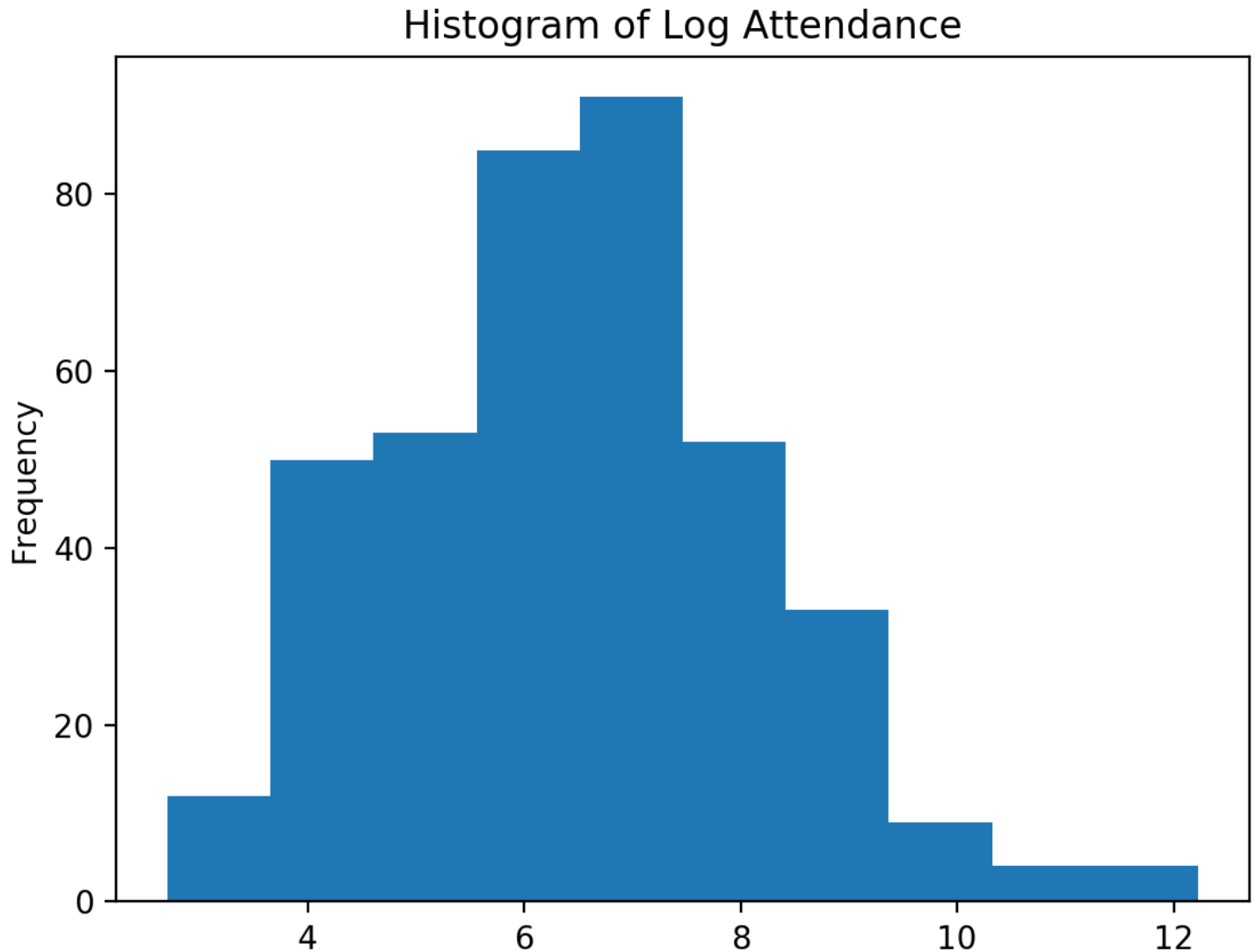
```
>>> (df_train.event_location_park.
...      value_counts(dropna=False).head())
NaN                  364
Magnuson Park          8
Gas Works Park         5
Occidental Park        3
Greenlake Park         2
Name: event_location_park, dtype: int64
```

EDA – attendance

- Estimated Attendance: few missing, right-skewed

```
>>> df_train.attendance.isnull().sum()
3
>>> x = df_train.attendance
>>> x.plot(kind='hist',
...     title='Histogram of Attendance')
>>> np.log(x).plot(kind='hist',
...     title='Histogram of Log Attendance')
```


Histogram of Attendance
Frequency
350
300
250
200
150
100
50
0
0
25000
50000
75000
100000
125000
150000
175000
200000


Histogram of Log Attendance
Frequency
0
20
40
60
80
4
6
8
10
12

Help! My Transformer Doesn't Exist!

- Use a `FunctionTransformer`
  - Turns any function into a transformer
  - Works well for stateless transformations

```
>>> from sklearn.preprocessing import FunctionTransformer
>>> logger = FunctionTransformer(np.log1p)
>>> X_log = logger.transform(X)
```

Help! My Transformer Doesn't Exist!

- Or write your own Custom Transformer

```
>>> from sklearn.base import TransformerMixin
>>> class Log1pTransformer(TransformerMixin):
...
...     def fit(self, X, y=None):
...         return self
...
...     def transform(self, X):
...         Xlog = np.log1p(X)
...         return Xlog
```

Custom Transformer – One Hot Encoding

- Goal: Turn string column into set of dummy variables

```
>>> from sklearn.feature_extraction import DictVectorizer
>>> class DummyTransformer(TransformerMixin):
...
...     def __init__(self):
...         self.dv = None
...
```

Custom Transformer – One Hot Encoding

- Convert each row to map: column name -> value
- Then fit a `DictVectorizer`

```
...     def fit(self, X, y=None):
...         Xdict = X.to_dict('records')
...         self.dv = DictVectorizer(sparse=False)
...         self.dv.fit(Xdict)
...         return self
...
```

Custom Transformer – One Hot Encoding

- Convert to map and apply `DictVectorizer`
- Then convert back to `DataFrame`

```
...     def transform(self, X):
...         Xdict = X.to_dict('records')
...         Xt = self.dv.transform(Xdict)
...         cols = self.dv.get_feature_names()
...         Xdum = pd.DataFrame(Xt, index=X.index,
...             columns=cols)
...         return Xdum
```

pandas + Pipelines

`pandas` versus `scikit-learn`

`pandas` DataFrames

- Support many data types
- Allow missing data
- Labeled rows and columns

`scikit-learn` Models

- Expect all numeric features
- Can't handle nulls (usually)
- Cast to `numpy arrays`

Solution

- `Pipeline`s are already `pandas`-friendly
  - If component Transformers are
- Write custom Transformers that return `DataFrames`
  - Subset columns
  - Wrap existing Transformers
  - More complex logic

Custom Transformer – Subset Columns

```
>>> class ColumnExtractor(TransformerMixin):
...
...     def __init__(self, cols):
...         self.cols = cols
...
...     def fit(self, X, y=None):
...         return self
...
...     def transform(self, X):
...         Xcols = X[self.cols]
...         return Xcols
```

Custom Transformer – Standardization

```
>>> class DFStandardScaler(TransformerMixin):
...
...     def __init__(self):
...         self.ss = None
...
...     def fit(self, X, y=None):
...         self.ss = StandardScaler().fit(X)
...         return self
...
```

Custom Transformer – Standardization

```
...     def transform(self, X):
...         Xss = self.ss.transform(X)
...         Xscaled = pd.DataFrame(Xss, index=X.index,
...             columns=X.columns)
...         return Xscaled
```

Putting It All Together – Feature Engineering

- List features by transformation strategy

```
>>> CAT_FEATS = [
...     'permit_type', 'event_category',
...     'event_sub_category', 'event_location_park',
...     'event_location_neighborhood']
>>> NUM_FEATS = ['attendance']
```

Putting It All Together – Feature Engineering

- One hot encoding for categorical features

```
>>> pipeline = Pipeline([
...     ('features', DFFeatureUnion([
...         ('categoricals', Pipeline([
...             ('extract', ColumnExtractor(CAT_FEATS)),
...             ('dummy', DummyTransformer())
...         ])),
```

Putting It All Together – Feature Engineering

- Log attendance, then standardize everything

```
...             ('numerics', Pipeline([
...                 ('extract', ColumnExtractor(NUM_FEATS)),
...                 ('zero_fill', ZeroFillTransformer()),
...                 ('log', Log1pTransformer())
...             ])),
...         ])),
...         ('scale', DFStandardScaler())
... ])
```

Putting It All Together – A Better Model

- Apply `Pipeline`

```
>>> X_train = pipeline.fit_transform(df_train)
>>> X_test = pipeline.transform(df_test)
```

- Fit model

```
>>> model = LogisticRegression()
>>> model.fit(X_train, y_train)
```

Putting It All Together – A Better Model

- Predict on test data

```
>>> p_pred_test = model.predict_proba(X_test)[:, 1]
```

- Measure performance on test data
  - Recall: first model had AUC = 0.57

```
>>> auc_test = roc_auc_score(y_test, p_pred_test) # 0.71
```

Links

- Dataset
  - City of Seattle Open Data portal
  - https://data.seattle.gov/Permitting/Special-Events-Permits/dm95-f8w5
- Inspiration
  - Blog post by Zac Stewart
  - http://zacstewart.com/2014/08/05/pipelines-of-featureunions-of-pipelines.html
- Content
  - This presentation and supporting code
  - https://github.com/jem1031/pandas-pipelines-custom-transformers

Questions?

Appendix

One Hot Encoding – Desired Properties

- Fit encoding on training data, then apply to test data
- Gracefully handle new levels in test data
- Transform a `DataFrame` to a (wider) `DataFrame`

One Hot Encoding – Existing Solutions

- `pandas.get_dummies`
  - can't easily apply encoding to new dataset
- `sklearn.preprocessing.LabelEncoder` & `OneHotEncoder`
  - errors out if new levels in test data
- `sklearn.feature_extraction.DictVectorizer`
  - input is list of `dicts`, output is `numpy array`

Custom Transformer – Missing to Zeroes

```
>>> class ZeroFillTransformer(TransformerMixin):
...
...     def fit(self, X, y=None):
...         return self
...
...     def transform(self, X):
...         Xz = X.fillna(value=0)
...         return Xz
```

Custom Transformer – Parallel Transformers

```
>>> from functools import reduce
>>> class DFFeatureUnion(TransformerMixin):
...
...     def __init__(self, transformer_list):
...         self.transformer_list = transformer_list
...
...     def fit(self, X, y=None):
...         for (name, t) in self.transformer_list:
...             t.fit(X, y)
...         return self
...
```

Custom Transformer – Parallel Transformers

```
...     def transform(self, X):
...         Xts = [t.transform(X)
...             for _, t in self.transformer_list]
...         Xunion = reduce(lambda X1, X2:
...             pd.merge(X1, X2,
...             left_index=True, right_index=True), Xts)
...         return Xunion
```