End-to-End Machine Learning: Preparing Data for Machine Learning

Glenn Bruns
CSUMB

Learning outcomes

After this lecture you should be able to:

1. Explain the Scikit-Learn approach to data preparation
2. Write Python code to:
   - handle missing data
   - convert categorical data to numeric
   - scale data
3. Write data pipelines

Data preprocessing in R and Scikit-Learn

In R:

- Data usually stored in data frames
- Data preprocessing depends on machine learning algorithm
- Example: use scaling before cluster analysis
- Example: classification trees can handle categorical variables

In Scikit-Learn:

- Data always stored in Numpy matrix (not Pandas data frame)
- **Always** remove NA values, convert all data to numeric, and scale
- This is called "standardization of datasets" in Scikit-Learn.

Dealing with NaN values

Several approaches:

1. Remove rows that contain any NaN values
2. Remove attributes that contain any NaN value
3. Convert NaN values
   - for example, replace NaN values of an attribute with the median or mode of the attribute

Question: if you were to use 1 or 2, which one would you use?

Functions for NaN handling

Pandas: dropna(), drop(), fillna()

```
dat['Embarked'].fillna(dat['Embarked'].mode()[0], inplace=True)
```

The mode() function is handy, applies to Series and DataFrames.

```
df = pd.DataFrame({'A': [1,2,3,4,3,2,3,1],
                   'B': ["a","b","c","d","d","d","a","b"],
                   'C': [1.2, 1.4, 1.1, 1.2, 1.6, 1.1, 1.5, 1.8]})


df['A'].mode()
Out[159]:
0    3
dtype: int64

df.mode()
Out[160]:
     A    B    C
0  3.0    d  1.1
1  NaN  NaN  1.2
```

Functions for NaN handling

Scikit-Learn: methods of the ‘Imputer’ class

```
num_dat = dat[[‘Age’, ‘Fare’]]   # imputer handles only numeric data
imputer = Imputer(strategy="median")
imputer.fit(num_dat)
X = imputer.transform(num_dat)   # X is a Numpy array, not a DataFrame
num_dat = pd.DataFrame(X, columns=num_dat.columns)
```

Note: NaN is represented as np.nan, and Pandas treats None like np.nan

Imputer's fit and transform methods

You can apply 'fit' and 'transform' separately:

```
imputer = Imputer(strategy="median")
imputer.fit(num_dat)
imputer.statistics_
X = imputer.transform(num_dat)
```

Or apply them in one step:

```
imputer = Imputer(strategy="median")
X = imputer.fit_transform(num_dat)
```

'fit' is finding (and recording) the the median
'transform' is transforming the data

Data scaling

Recall: typical scaling methods

1. Unit interval scaling (0-1 scaling)
2. Z-score normalization

```
# must remove NaNs before scaling
imputer = Imputer(strategy="median")
X = imputer.fit_transform(dat[['Age', 'Fare']])

# Z-score normalization
scaler = StandardScaler()
num_dat_scaled = scaler.fit_transform(X)
```

StandardScalers have 'fit' and 'transform' methods, too.

Converting categorical to numeric data

Idea:

1. first convert data to integer categorical data
2. then replace with indicator variables (also known as "dummy variables", or "one-hot encoding")

```
# do both steps at once with sklearn's LabelBinarizer
encoder = LabelBinarizer()
pclass_1hot = encoder.fit_transform(dat['Pclass'])
```

LabelBinarizer objects also have 'fit' and 'transform' methods.

Note: this is more easily done with pandas using the get_dummies() method on data frames.

Automate data preprocessing

Recall:

Avoid manual preprocessing steps!

Support “redo analysis by the press of a button”

Pipelines

Data processing pipeline:

A programmed sequence of data processing steps, which includes:

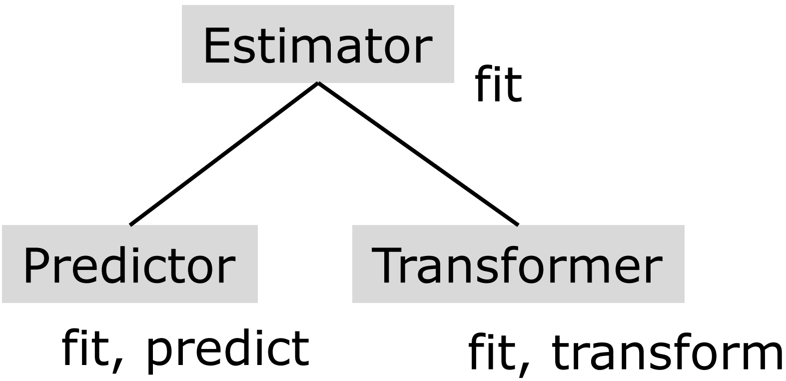
- data acquisition
- data cleaning and munging
- machine learning

Simple example: impute, then scale, then apply machine learning algorithm.

The Scikit-Learn API that explicitly supports pipelines.

Recommended reading:
Buitinck et al, *API design for machine learning software: experiences from the Scikit-Learn project*, 2013.

Scikit-Learn Interfaces



fit: X,y → ()
learn model from training data

predict: X → predictions
make predictions from data

transform: X → X
transform data

Examples:

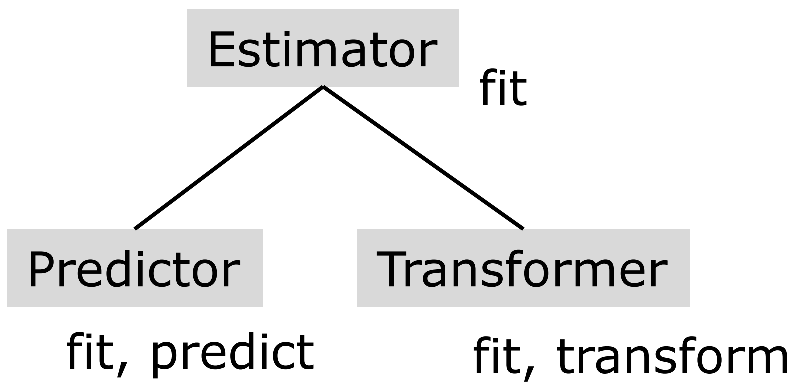
- Preprocessing and feature selection: Transformers.

```
imputer.fit(X)
X1 = imputer.transform(X)
```

- Supervised learning: Predictors.

```
log_regr.fit(X_train,y_train)
y_pred =
     log_regr.predict(X_test)
```

Pipelines in Scikit-Learn



Pipeline:

A list (possibly empty) of Transformers followed by an Estimator.

Used as a Predictor or Transformer.

```
# simple preprocessing pipeline
pipe = Pipeline([
        ("remove_nas", Imputer(strategy="median")),
        ("z-scaling",  StandardScaler())
        ])

pipe.fit_transform(dat[['Age', 'Fare']])
```

Pipelines in Scikit-Learn

```
# simple preprocessing pipeline
pipe = Pipeline([
        ("remove_nas", Imputer(strategy="median")),
        ("z-scaling",  StandardScaler())
        ])

dat_transformed = pipe.fit_transform(dat[['Age', 'Fare']])
```

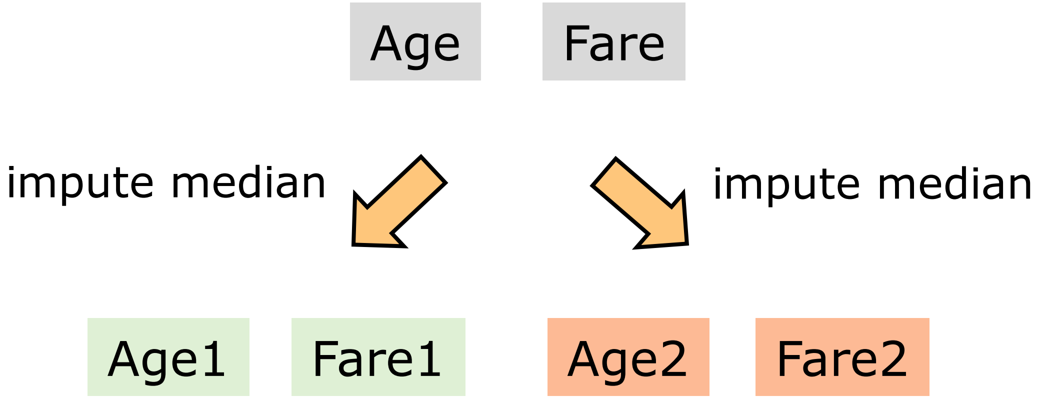
The fit_transform method does this: (let X be the data)

- imputer.fit(X)
- X_imputed = imputer.transform(X)
- scaler.fit(X_imputed)
- X_scaled = scaler.transform(X)

FeatureUnion

Situations that often arise:

- need to use different preprocessing for numeric and categorical variables
- from an initial set of features, create many derived features



FeatureUnion

FeatureUnion:

A list of transformers. The fit operation applies the transformers "in parallel" and concatenates their outputs.

```
# use two different methods to impute NaN value
union = FeatureUnion([
        ("impute_median", Imputer(strategy="median")),
        ("impute_mean",   Imputer(strategy="mean"))
        ])

dat1 = union.fit_transform(dat[['Age', 'Fare']])
```

Pipeline + FeatureUnion

You can create complex workflows by combining pipelines and feature unions.

What is this example doing?

```
from sklearn.pipeline import FeatureUnion, Pipeline
from sklearn.decomposition import PCA, KernelPCA
from sklearn.featureselection import SelectKBest

union = FeatureUnion([(“pca”, PCA()),
                      (“kpca”, KernelPCA(kernel=“rbf”))])
Pipeline([(“feat_union”, union),
          (“feat_sel”,   SelectKBest(k=10)),
          (“log_reg”,    LogisticRegression(penalty=“12”))
        ]).fit(X_train, y_train).predict(X_test)
```

source: Buitinck et al, API design for machine learning software, 2013

Pipeline + FeatureUnion, example 2

```
num_attribs = list(housing_num)
cat_attribs = [“ocean_proximity”]

num_pipeline = Pipeline([
     (‘selector’, DataFrameSelector(num_attribs)),
     (‘imputer’,  Imputer(strategy=“median”)),
     (‘attribs_adder’, CombinedAttributesAdder()),
     (‘std_scaler’, StandardScaler()),
  ])

cat_pipeline = Pipeline([
     (‘selector’, DataFrameSelector(cat_attribs)),
     (‘label_binarizer’,  LabelBinarizer()),
  ])

full_pipeline = FeatureUnion(transformer_list=[
     (‘num_pipeline’, num_pipeline),
     (‘cat_pipeline’, cat_pipeline),
  ])
```

source: Geron text

Summary

- Pandas and Scikit-Learn each include preprocessing functions
- Scikit-Learn supports the idea of a standard API for all kinds of data processing:
  - Estimator, Transformer, Predictor
- Using Scikit-Learn's Pipeline and FeatureUnion, you can build complex workflows