Preprocessing data

SUPERVISED LEARNING WITH SCIKIT-LEARN

Andreas Müller
Core developer, scikit-learn

Dealing with categorical features

- Scikit-learn will not accept categorical features by default
- Need to encode categorical features numerically
- Convert to 'dummy variables'
    - 0: Observation was NOT that category
    - 1: Observation was that category

Dummy variables

Origin
US
Europe
Asia

Dummy variables

Origin
US
Europe
Asia

origin_Asia	origin_Europe	origin_US
0	0	1
0	1	0
1	0	0

Dummy variables

Origin
US
Europe
Asia

origin_Asia	origin_US
0	1
0	0
1	0

Dealing with categorical features in Python

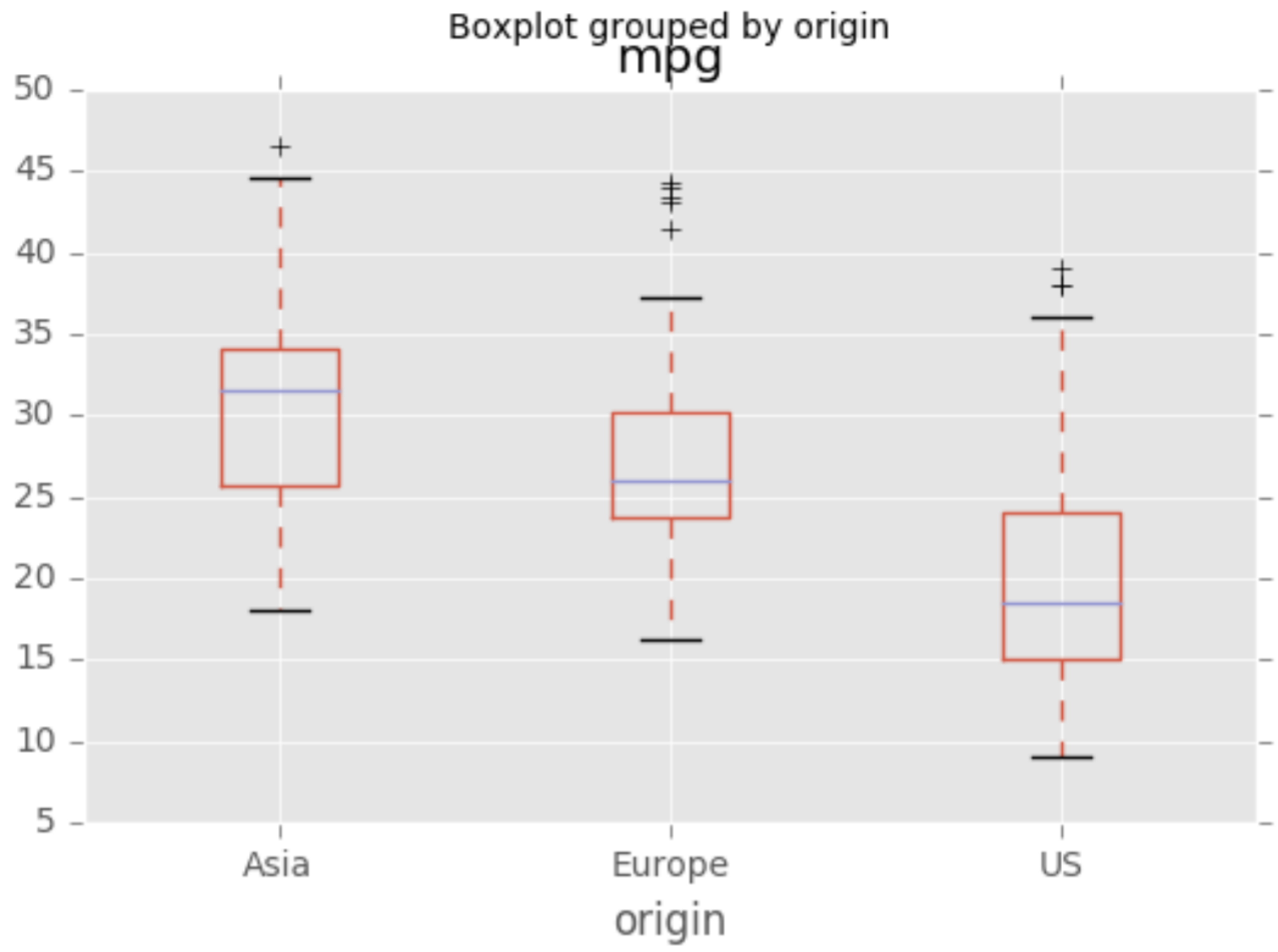
- scikit-learn: OneHotEncoder()
- pandas: get_dummies()

Automobile dataset

- mpg: Target Variable
- Origin: Categorical Feature

	mpg	displ	hp	weight	accel	origin	size
0	18.0	250.0	88	3139	14.5	US	15.0
1	9.0	304.0	193	4732	18.5	US	20.0
2	36.1	91.0	60	1800	16.4	Asia	10.0
3	18.5	250.0	98	3525	19.0	US	15.0
4	34.3	97.0	78	2188	15.8	Europe	10.0

EDA w/ categorical feature



Encoding dummy variables

```
import pandas as pd
df = pd.read_csv('auto.csv')
df_origin = pd.get_dummies(df)
print(df_origin.head())
```

```
    mpg  displ   hp  weight  accel  size  origin_Asia  origin_Europe  \\
0  18.0  250.0   88    3139   14.5  15.0            0              0
1   9.0  304.0  193    4732   18.5  20.0            0              0
2  36.1   91.0   60    1800   16.4  10.0            1              0
3  18.5  250.0   98    3525   19.0  15.0            0              0
4  34.3   97.0   78    2188   15.8  10.0            0              1
   origin_US
0          1
1          1
2          0
3          1
4          0
```

Encoding dummy variables

```
df_origin = df_origin.drop('origin_Asia', axis=1)
print(df_origin.head())
```

```
    mpg  displ   hp  weight  accel  size  origin_Europe  origin_US
0  18.0  250.0   88    3139   14.5  15.0              0          1
1   9.0  304.0  193    4732   18.5  20.0              0          1
2  36.1   91.0   60    1800   16.4  10.0              0          0
3  18.5  250.0   98    3525   19.0  15.0              0          1
4  34.3   97.0   78    2188   15.8  10.0              1          0
```

Linear regression with dummy variables

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
```

```
ridge = Ridge(alpha=0.5, normalize=True).fit(X_train,
                                              y_train)
```

```
ridge.score(X_test, y_test)
```

```
0.719064519022
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Handling missing data

SUPERVISED LEARNING WITH SCIKIT-LEARN

Hugo Bowne-Anderson
Data Scientist, DataCamp

PIMA Indians dataset

```
df = pd.read_csv('diabetes.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
pregnancies    768 non-null int64
glucose        768 non-null int64
diastolic      768 non-null int64
triceps        768 non-null int64
insulin        768 non-null int64
bmi            768 non-null float64
dpf            768 non-null float64
age            768 non-null int64
diabetes       768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

PIMA Indians dataset

```
print(df.head())
```

```
   pregnancies  glucose  diastolic  triceps  insulin   bmi    dpf  age  \\
0            6      148         72       35        0  33.6  0.627   50
1            1       85         66       29        0  26.6  0.351   31
2            8      183         64        0        0  23.3  0.672   32
3            1       89         66       23       94  28.1  0.167   21
4            0      137         40       35      168  43.1  2.288   33
   diabetes
0        1
1        0
2        1
3        0
4        1
```

Dropping missing data

```
df.insulin.replace(0, np.nan, inplace=True)
df.triceps.replace(0, np.nan, inplace=True)
df.bmi.replace(0, np.nan, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
pregnancies    768 non-null int64
glucose        768 non-null int64
diastolic      768 non-null int64
triceps        541 non-null float64
insulin        394 non-null float64
bmi            757 non-null float64
dpf            768 non-null float64
age            768 non-null int64
diabetes       768 non-null int64
dtypes: float64(4), int64(5)
memory usage: 54.1 KB
```

Dropping missing data

```
df = df.dropna()
df.shape
```

```
(393, 9)
```

Imputing missing data

- Making an educated guess about the missing values
- Example: Using the mean of the non-missing entries

```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0
imp.fit(X)
X = imp.transform(X)
```

Imputing within a pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
logreg = LogisticRegression()
steps = [('imputation', imp),
             ('logistic_regression', logreg)]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.3, random_state=42)
```

Imputing within a pipeline

```
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
pipeline.score(X_test, y_test)
```

```
0.7532467532467532
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Centering and scaling

SUPERVISED LEARNING WITH SCIKIT-LEARN

Hugo Bowne-Anderson
Data Scientist, DataCamp

Why scale your data?

```
print(df.describe())
```

```
      fixed acidity  free sulfur dioxide  total sulfur dioxide      density  \\
count   1599.000000          1599.000000           1599.000000  1599.000000
mean       8.319637            15.874922             46.467792     0.996747
std        1.741096            10.460157             32.895324     0.001887
min        4.600000             1.000000              6.000000     0.990070
25%        7.100000             7.000000             22.000000     0.995600
50%        7.900000            14.000000             38.000000     0.996750
75%        9.200000            21.000000             62.000000     0.997835
max       15.900000            72.000000            289.000000     1.003690
                 pH    sulphates      alcohol      quality
count   1599.000000  1599.000000  1599.000000  1599.000000
mean       3.311113     0.658149    10.422983     0.465291
std        0.154386     0.169507     1.065668     0.498950
min        2.740000     0.330000     8.400000     0.000000
25%        3.210000     0.550000     9.500000     0.000000
50%        3.310000     0.620000    10.200000     0.000000
75%        3.400000     0.730000    11.100000     1.000000
max        4.010000     2.000000    14.900000     1.000000
```

Why scale your data?

- Many models use some form of distance to inform them
- Features on larger scales can unduly influence the model
- Example: k-NN uses distance explicitly when making predictions
- We want features to be on a similar scale
- Normalizing (or scaling and centering)

Ways to normalize your data

- Standardization: Subtract the mean and divide by variance
- All features are centered around zero and have variance one
- Can also subtract the minimum and divide by the range
- Minimum zero and maximum one
- Can also normalize so the data ranges from -1 to +1
- See scikit-learn docs for further details

Scaling in scikit-learn

```python
from sklearn.preprocessing import scale
X_scaled = scale(X)
```

```python
np.mean(X), np.std(X)
```

```
(8.13421922452, 16.7265339794)
```

```python
np.mean(X_scaled), np.std(X_scaled)
```

```
(2.54662653149e-15, 1.0)
```

Scaling in a pipeline

```
from sklearn.preprocessing import StandardScaler
steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=21)
knn_scaled = pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
0.956
```

```
knn_unscaled = KNeighborsClassifier().fit(X_train, y_train)
knn_unscaled.score(X_test, y_test)
```

```
0.928
```

CV and scaling in a pipeline

```
steps = [('scaler', StandardScaler()),
        (('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
parameters = {knn__n_neighbors=np.arange(1, 50)}
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=21)
cv = GridSearchCV(pipeline, param_grid=parameters)
cv.fit(X_train, y_train)
y_pred = cv.predict(X_test)
```

Scaling and CV in a pipeline

```
print(cv.best_params_)
```

```
{'knn__n_neighbors': 41}
```

```
print(cv.score(X_test, y_test))
```

```
0.956
```

```
print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support
          0       0.97      0.90      0.93        39
          1       0.95      0.99      0.97        75
avg / total       0.96      0.96      0.96       114
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

Final thoughts

SUPERVISED LEARNING WITH SCIKIT-LEARN

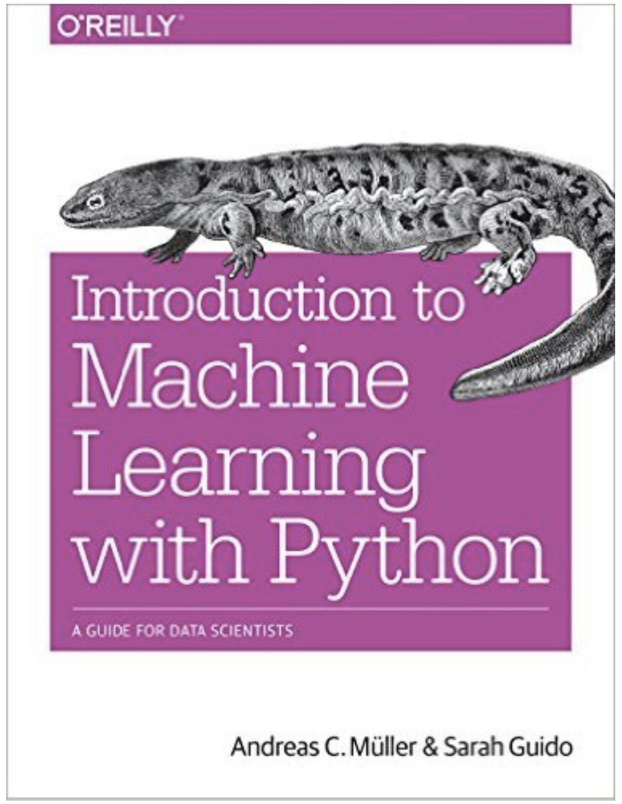
Hugo and Andy
Data Scientists

What you've learned

- Using machine learning techniques to build predictive models
- For both regression and classification problems
- With real-world data
- Underfitting and overfitting
- Test-train split
- Cross-validation
- Grid search

What you’ve learned

- Regularization, lasso and ridge regression
- Data preprocessing
- For more: Check out the scikit-learn documentation



Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN