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()

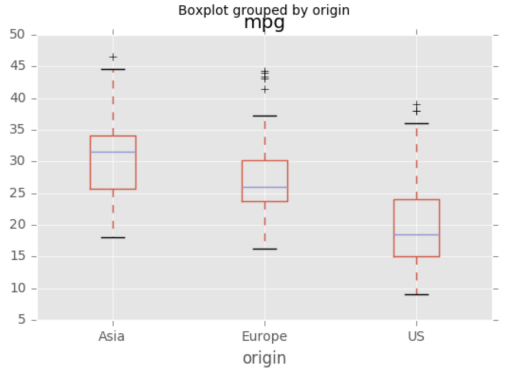
1.
sklearn.preprocessing.OneHotEncoder(categories='auto', drop=None, sparse=True, dtype=<class 'numpy.float64'>, handle_unknown='error')
Encode categorical features as a one-hot numeric array.
"drop" ('first' or an array-like of shape (n_features,), default=None) specifies a methodology to use to drop one of the categories per feature. This is useful in situations where perfectly collinear features cause problems, such as when feeding the resulting data into a neural network or an unregularized regression. It could possibly be None that retains all features (the default); or 'first' that drops the first category in each feature. If only one category is present, the feature will be dropped entirely. In addition to the above two possibilities, it could also be an array, for example, drop[i] is the category in feature X[:, i] that should be dropped.

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



2.
Categorical omega is a method to calculate coefficient omega for categorical items. When item scores are ordered categorical, categorical omega can be computed based on the parameter estimates from a factor analysis model using frequentist estimators such as diagonally weighted least squares.

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
```

3.
pandas.get_dummies(data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False, dtype=None) → 'DataFrame'
Convert categorical variable into dummy/indicator variables.
"drop_first" (bool, default False) decide whether to get k-1 dummies out of k categorical levels by removing the first level.

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
```

4.
DataFrame.replace(self, to_replace=None, value=None, inplace=False, limit=None, regex=False, method='pad')
Replace values given in to_replace with value.
"to_replace" (str, regex, list, dict, Series, int, float, or None) decide how to find the values that will be replaced.
"value" (scalar, dict, list, str, regex, default None) is the value to replace any values matching to_replace with. For a DataFrame a dict of values can be used to specify which value to use for each column (columns not in the dict will not be filled). Regular expressions, strings, and lists or dicts of such objects are also allowed.
"inplace" (bool, default False) a boolean, if it is True, in place. Note: this will modify any other views on this object (e.g., a column from a DataFrame). Returns the caller if this is True.

Dropping missing data

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

```
(393, 9)
```

5.
DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False)
Remove missing values.
"axis" ({0 or 'index', 1 or 'columns'}, default 0) determines if rows or columns which contain missing values are removed. When the assignment of "axis" is 0, or 'index', drop the rows which contain the missing values. When the assignment of "axis" is 1, or 'columns', drop the columns which contain the missing value.
"how" ({'any', 'all'}, default 'any') determines if row or column is removed from DataFrame, when we have at least one NA or all NA. When the assignment of "how" is 'any', means if any NA values are present, drop that row or column. When the assignment of "how" is 'all', means if all values are NA, drop that row or column.

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)
```

6.
For various reasons, many real-world datasets contain missing values, often encoded as blanks, NaNs, or other placeholders. Such datasets, however, are incompatible with scikit-learn estimators, which assume that all values in an array are numerical and that all have and hold meaning. A basic strategy to use incomplete datasets is to discard entire rows and/or columns containing missing values. However, this comes at the price of losing data, which may be valuable (even though incomplete). A better strategy is to impute the missing values, i.e., to infer them from the known part of the data.
The scikit-learn library provided the Imputer() pre-processing class that can be used to replace missing values until the version 0.16.1.

7.
sklearn.preprocessing.Imputer(missing_values='NaN', strategy='mean', axis=0, verbose=0, copy=True)
Imputation transformer for completing missing values.
"missing_values" (integer or "NaN", optional (default="NaN")) means the placeholder for the missing values. All occurrences of missing_values will be imputed. For missing values encoded as np.nan, use the string value "NaN".
"strategy" (string, optional (default=" mean")) means the imputation strategy. If its assignment is "mean", then replace missing values using the mean along the axis. If its assignment is "median", then replace missing values using the median along the axis. If its assignment is "most_frequent", then replace missing using the most frequent value along the axis.
"axis" (integer, optional (default=0)) means the axis along which to impute. If axis=0, then impute along columns; if axis=1, then impute along rows.

8.
transform(X)
Impute all missing values in X.
"X" {{array-like, sparse matrix}, shape = [n_samples, n_features]} is the input data to complete.

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)
```

9.
Pipeline can be used to chain multiple estimators into one. This is useful as there is often a fixed sequence of steps in processing the data, for example, feature selection, normalization, and classification.

10.
sklearn.pipeline.Pipeline(steps, memory=None, verbose=False)
The pipeline of transforms with a final estimator.
"steps" (list) means the list of (name, transform) tuples (implementing fit/transform) that are chained, in the order in which they are chained, with the last object an estimator.

Imputing within a pipeline

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

```
0.75324675324675328
```

Let's practice!

SUPERVISED LEARNING WITH SCIKIT-LEARN

DataCamp

Centering and scaling

SUPERVISED LEARNING WITH SCIKIT-LEARN

Hugo Bowne-Anderson
Data Scientist, DataCamp

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

11.
Standardization (Z-score Normalization) is a transformation that centers the data by removing the mean value of each feature and then scale it by dividing (non-constant) features by their standard deviation.

12.
Feature scaling through standardization can be an important preprocessing step for many machine learning algorithms. After standardizing data, the mean will be zero and the standard deviation one. That means standardization involves rescaling the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.

13.
Rescaling (min-max normalization) is an alternative standardization, which is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. The motivation to use this scaling include robustness to very small standard deviations of features and preserving zero entries in sparse data.

14.
Rescaling is a technique that rescales a feature or observation value with a distribution value between 0 and 1.

15. Mean normalization scales the value range to [-1, 1], and the mean of the data is close to 0. Both mean normalization and standardization move the data distribution center to the origin that is called zero-centered.
However, mean normalization does not change the shape of the data distribution, while standardization makes the distribution of sample data approximate to a certain distribution (usually a normal distribution).

Scaling in scikit-learn

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

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

```
(8.13421922452, 16.7265339794)
```

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

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

16.
Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn. They might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.
The function scale provides a quick and easy way to perform this operation on a single array-like dataset.

17.
sklearn.preprocessing.scale(X, axis=0, with_mean=True, with_std=True, copy=True)
Standardize a dataset along any axis
"X" ({array-like, sparse matrix}) is the data to center and scale.

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
```

18.
The preprocessing module further provides a utility class StandardScaler that implements the Transformer API to compute the mean and standard deviation on a training set so as to be able to later reapply the same transformation on the testing set. This class is hence suitable for use in the early steps of a sklearn.pipeline.Pipeline.

19.
sklearn.preprocessing.StandardScaler(copy=True, with_mean=True, with_std=True)
Standardize features by removing the mean and scaling to unit variance
"copy" (boolean, optional, default True) s boolean. If it is False, try to avoid a copy and do in place scaling instead.
"with_mean" (boolean, True by default) is boolean. If it is True, center the data before scaling.
"with_std" (boolean, True by default) is boolean. If it is True, scale the data to unit variance (or equivalently, unit standard deviation).

20.
sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)
Accuracy classification score.
"y_true" (1d array-like, or label indicator array / sparse matrix) means the ground truth (correct) labels.
"y_pred" (1d array-like, or label indicator array / sparse matrix) means the predicted labels, as returned by a classifier.

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)
```

21.
The syntax to define a parameter grid for a pipeline is to specify for each parameter the step name, followed by __ (a double underscore), followed by the parameter name. For example, to search over the C parameter of SVC, it is necessary to use "svm__C" as the key in the parameter grid dictionary, and similarly, for gamma, use "svm__gamma".

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

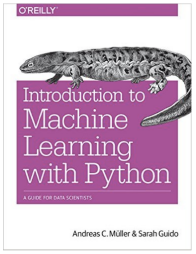
Hugo and Andy
Data Scientists

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