



Lisbon School
of Economics
& Management
Universidade de Lisboa



scikit-learn

Carlos J. Costa

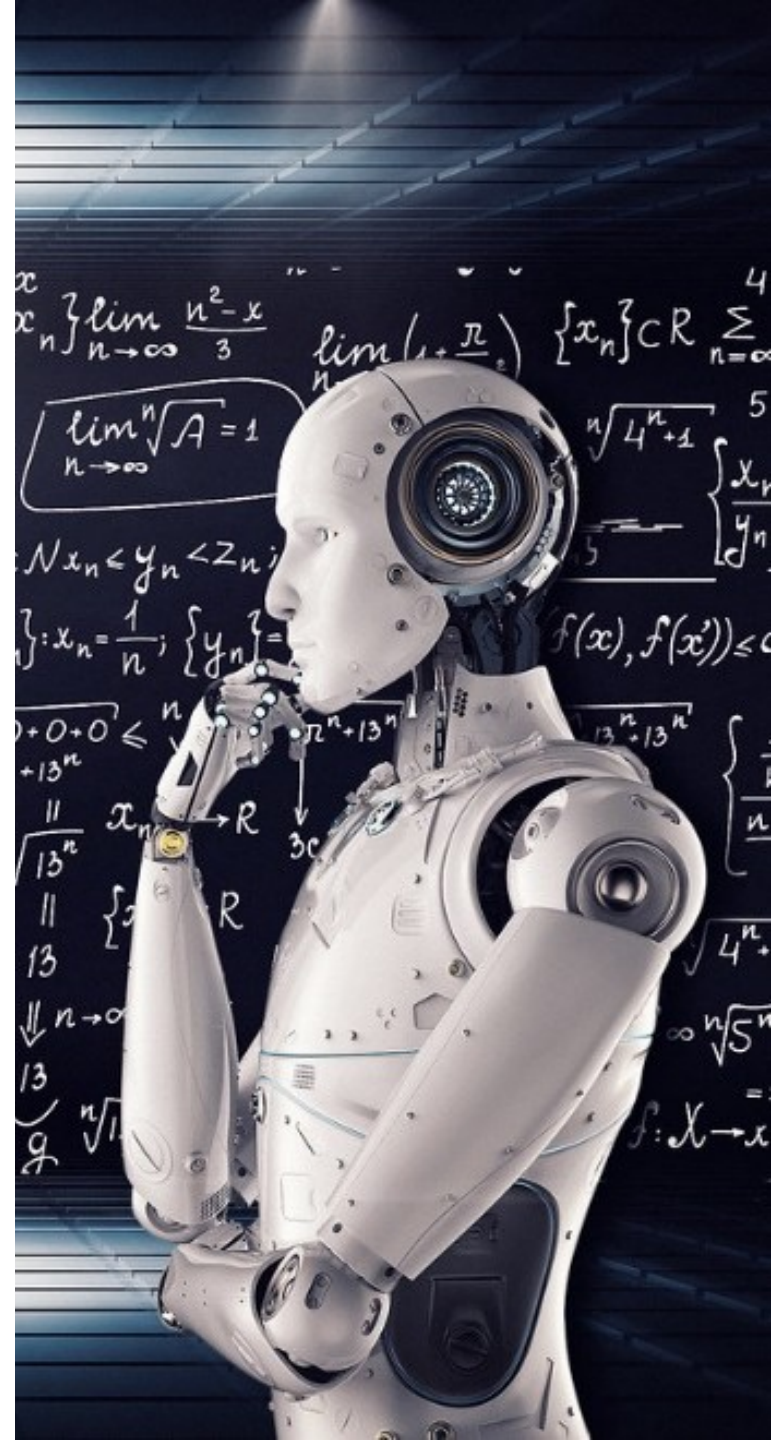


Learning Goals

- Distinguish between supervised and unsupervised learning
- Pre-processing data using scikit-learn
- Creating and fitting model using scikit-learn
- Understand how to use pipeline in scikit-learn

Machine Learning

- subset of artificial intelligence
- enable systems to learn patterns from data
- improve from experience





Scikit-Learn

- tools for predictive data analysis
- reusable in various contexts
- built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license
- <https://scikit-learn.org>
- Refence:
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12, 2825-2830.

Machine Learning

Learning problems fall into a few categories:

- supervised learning
- unsupervised learning

	<i>Supervised Learning</i>	<i>Unsupervised Learning</i>
<i>Discrete</i>	classification or categorization	clustering
<i>Continuous</i>	regression	dimensionality reduction

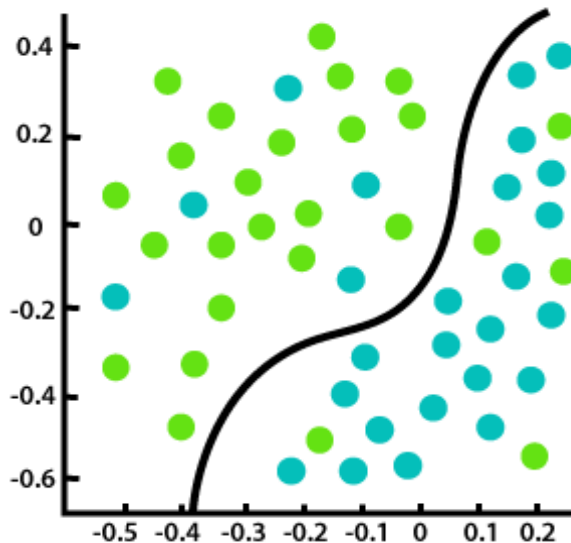
supervised learning

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...



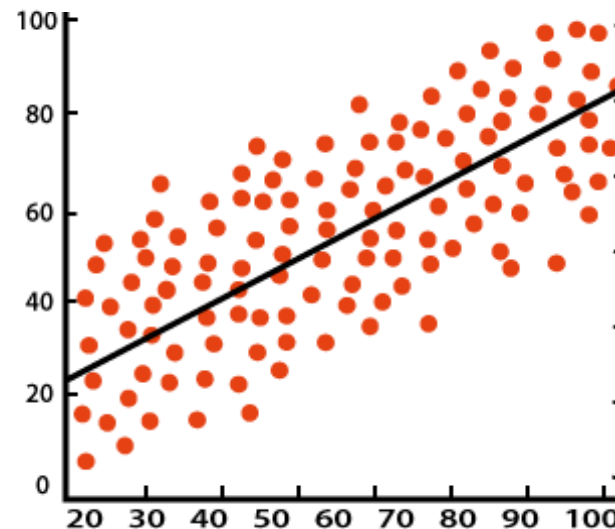
Classification

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, nearest neighbors, random forest, and more...



Regression

unsupervised learning

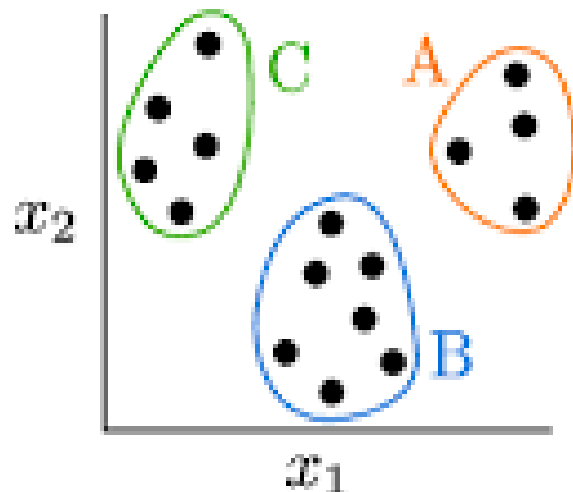
Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...

Clustering



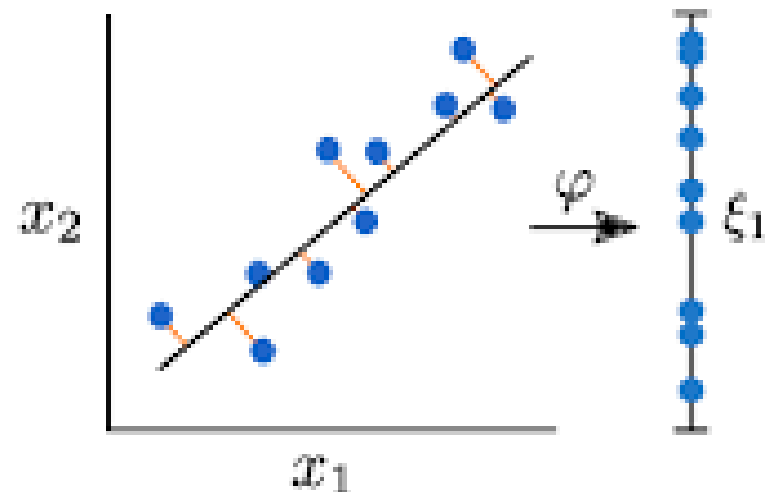
Dimensionality reduction

Reducing the number of random variables to consider.

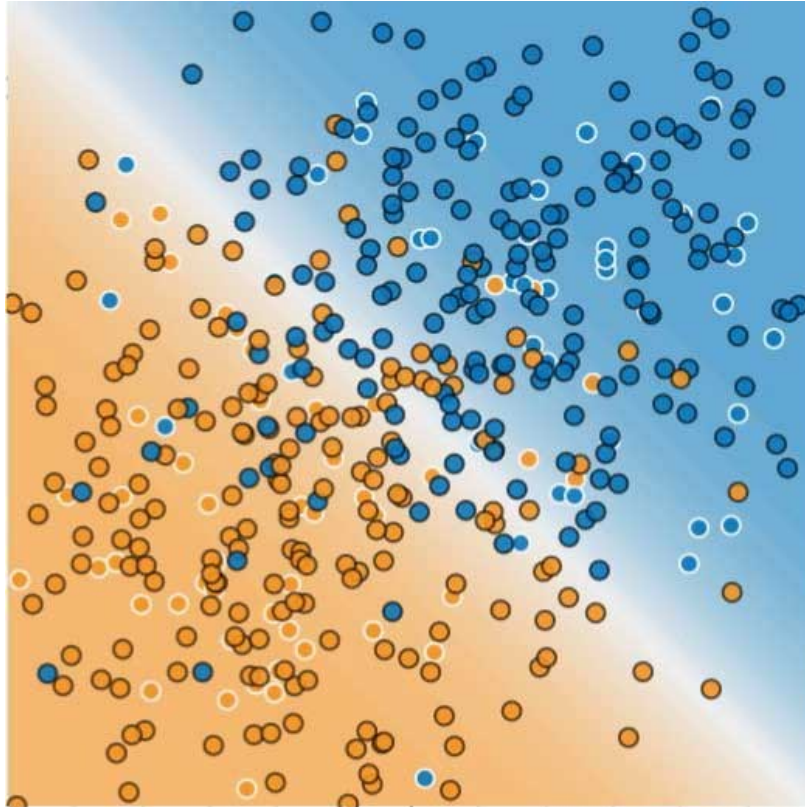
Applications: Visualization, Increased efficiency

Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...

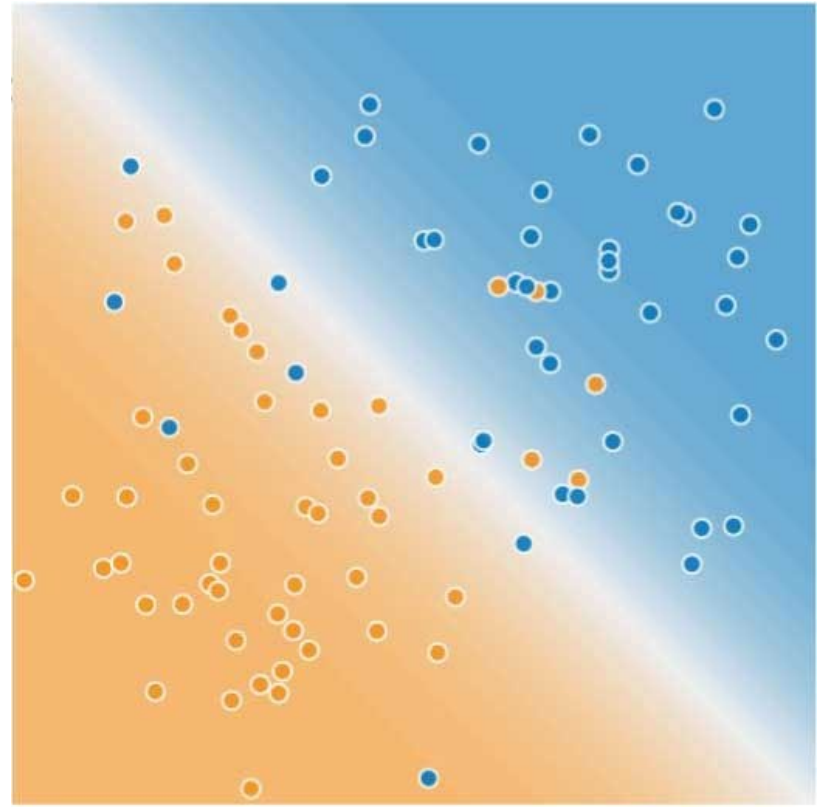
Dimensionality Reduction



Training set and testing set



Training Data



Test Data

Learning and predicting

```
import sklearn.linear_model as lm
import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
```

```
# Create linear regression object
reg = lm.LinearRegression()
# Train the model
results = reg.fit(df[['x']], df['y'])
```

```
results.score(df[['x']], df['y'])
```

```
0.7714285714285715
```

```
results.coef_
```

```
array([0.9])
```

```
results.intercept_
```

```
0.5
```

```
results.predict([[3]])
```

```
array([3.2])
```

Learning and predicting

```
import sklearn.linear_model as lm
import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
```

```
# Create linear regression object
reg = lm.LinearRegression()
# Train the model
results = reg.fit(df[['x']], df['y'])
```

```
results.score(df[['x']], df['y'])
```

```
0.7714285714285715
```

```
results.coef_
```

```
array([0.9])
```

```
results.intercept_
```

```
0.5
```

```
results.predict([[3]])
```

```
array([3.2])
```

Learning and predicting

```
import sklearn.linear_model as lm
import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
```

```
# Create linear regression object
reg = lm.LinearRegression()
# Train the model
results = reg.fit(df[['x']], df['y'])
```

```
results.score(df[['x']], df['y'])
```

```
0.7714285714285715
```

```
results.coef_
```

```
array([0.9])
```

```
results.intercept_
```

```
0.5
```

```
results.predict([[3]])
```

```
array([3.2])
```

```
df[['x']].shape
```

```
(16,)
```

```
df[['x']].shape
```

```
(16, 1)
```

Learning and predicting

```
import sklearn.linear_model as lm
import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
```

```
# Create linear regression object
reg = lm.LinearRegression()
# Train the model
results = reg.fit(df[['x']], df[['y']])
```

```
results
```

```
0.771
```

```
results
```

```
array
```

```
results
```

```
0.5
```

```
# Create linear regression object
reg = lm.LinearRegression()
# Train the model
results = reg.fit(df[['x']], df[['y']])
```

```
results.predict([[3]])
```

```
array([3.2])
```

Learning and predicting

```
import sklearn.linear_model as lm
import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])

# Create linear regression object
reg = lm.LinearRegression()
# Train the model
results = reg.fit(df[['x']], df[['y']])

results.coef_
array([0.9])

results.coef_
array([0.9])

results.intercept_
0.5

results.predict([[3]])
array([3.2])
```

Learning and predicting

```
import sklearn.linear_model as lm

import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
```

```
# Split the data into training/testing sets
x_train = x[:-6]
x_test = x[-6:]
# Split the targets into training/testing sets
y_train = y[:-6]
y_test = y[-6:]
```

```
[ [3,2], [1,2], [2,2], [4,4], [3,3], [1,2], [2,3], [4,4], [3,3], [1,1],  [2,2], [4,5], [3,4], [1,1], [2,2], [4,4] ]
```

```
# Create linear regression object
regr = lm.LinearRegression()
# Train the model using the training sets
regr.fit(x_train, y_train)
# Make predictions using the testing set
y_pred = regr.predict(x_test)
# The coefficients
print('Coefficients: \n', regr.coef_)
```

```
Coefficients:
[0.69354839]
```

Learning and predicting

```
import sklearn.linear_model as lm

import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
```

```
# Split the data into training/testing sets
x_train = x[:-6]
x_test = x[-6:]
# Split the targets into training/testing sets
y_train = y[:-6]
y_test = y[-6:]
```

Train

`[ [3,2], [1,2], [2,2], [4,4], [3,3], [1,2], [2,3], [4,4], [3,3], [1,1] ]`

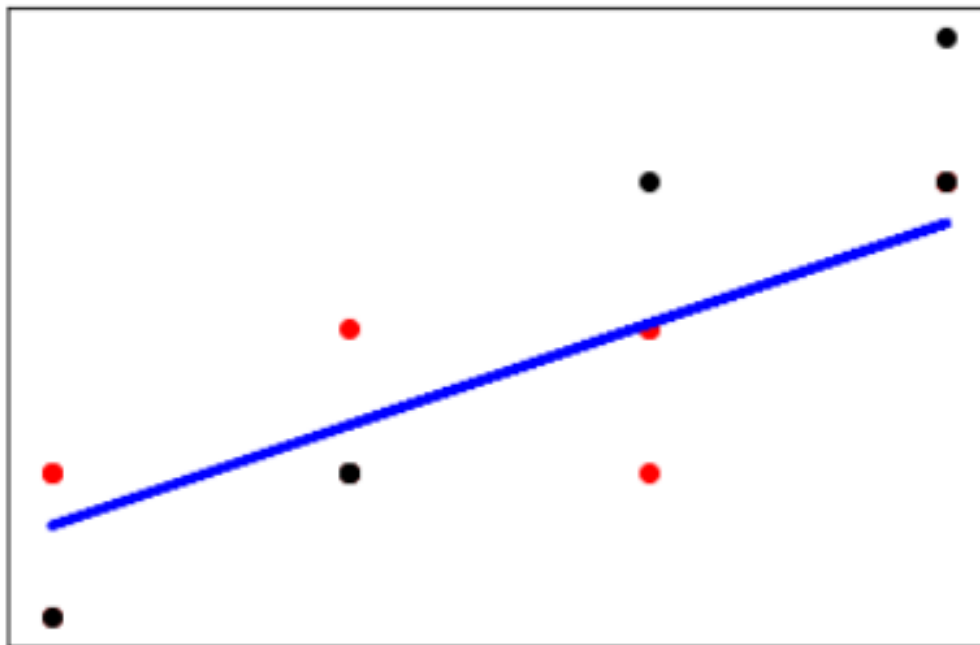
`[ [2,2], [4,5], [3,4], [1,1], [2,2], [4,4] ]`

```
# Create linear regression object
regr = lm.LinearRegression()
# Train the model using the training sets
regr.fit(x_train, y_train)
# Make predictions using the testing set
y_pred = regr.predict(x_test)
# The coefficients
print('Coefficients: \n', regr.coef_)
```

Test

```
Coefficients:
[0.69354839]
```

```
import matplotlib.pyplot as plt
# Plot outputs
plt.scatter(x_train, y_train, color='red')
plt.scatter(x_test, y_test, color='black')
plt.plot(x_test, y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

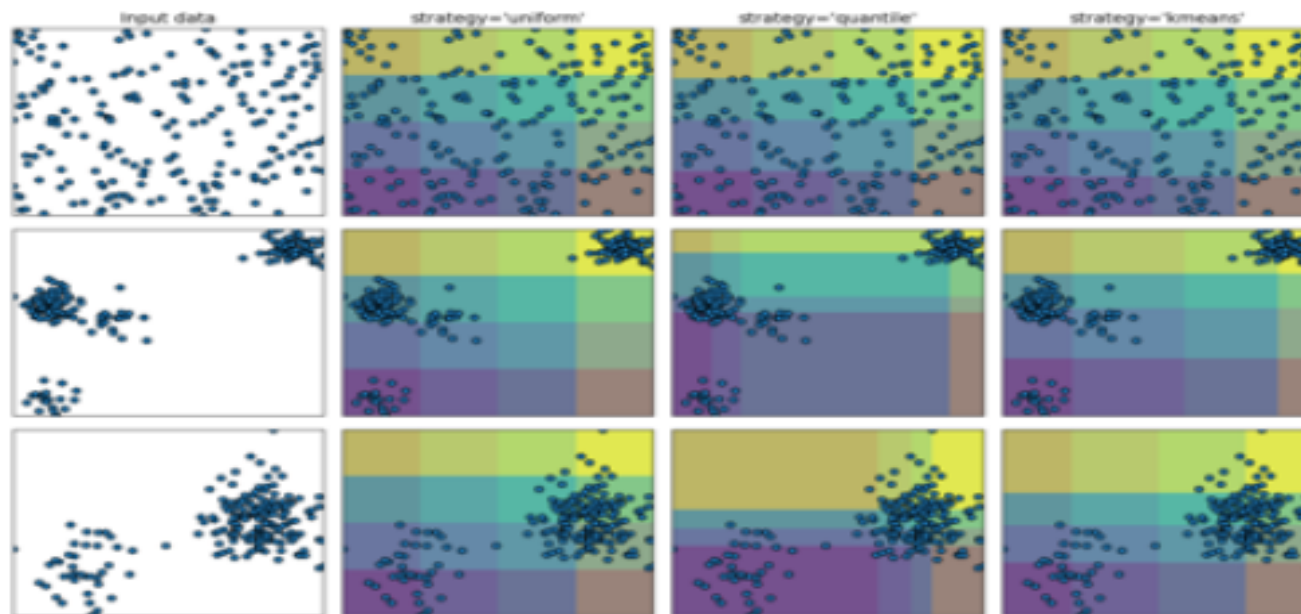


Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.

Algorithms: preprocessing, feature extraction, and more...



Inconsistent pre-processing

```
import sklearn.linear_model as lm
import sklearn.model_selection as ms
import sklearn.metrics as metrics
import sklearn.preprocessing as prep

import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
X=df[['x']]
y=df[['y']]
```

```
X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size=0.4, random_state=50)
```

```
scaler = prep.StandardScaler()
X_train_transformed = scaler.fit_transform(X_train)
model = lm.LinearRegression().fit(X_train_transformed, y_train)
metrics.r2_score(y_test, model.predict(X_test))
```

-2.1041861708478207

```
scaler = prep.StandardScaler()
X_train_transformed = scaler.fit_transform(X_train)
model = lm.LinearRegression().fit(X_train_transformed, y_train)
X_test_transformed = scaler.transform(X_test)
metrics.r2_score(y_test, model.predict(X_test_transformed))
```

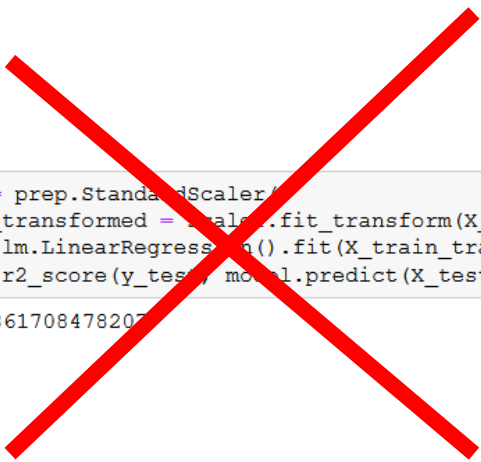
0.8150675457605236

Inconsistent pre-processing

```
import sklearn.linear_model as lm
import sklearn.model_selection as ms
import sklearn.metrics as metrics
import sklearn.preprocessing as prep

import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
X=df[['x']]
y=df['y']
```

```
X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size=0.4, random_state=50)
```



```
scaler = prep.StandardScaler()
X_train_transformed = scaler.fit_transform(X_train)
model = lm.LinearRegression().fit(X_train_transformed, y_train)
metrics.r2_score(y_test, model.predict(X_test))
```

-2.1041861708478207

```
scaler = prep.StandardScaler()
X_train_transformed = scaler.fit_transform(X_train)
model = lm.LinearRegression().fit(X_train_transformed, y_train)
X_test_transformed = scaler.transform(X_test)
metrics.r2_score(y_test, model.predict(X_test_transformed))
```

0.8150675457605236

Inconsistent pre-processing

```
import sklearn.pipeline as pipeline
import sklearn.linear_model as lm
import sklearn.model_selection as ms
import sklearn.metrics as metrics
import sklearn.preprocessing as prep
import pandas as pd
data=[[3,2],[1,2],[2,2],[4,4],[3,3],[1,2],[2,3],[4,4],[3,3],[1,1],[2,2],[4,5],[3,4],[1,1],[2,2],[4,4]]
df=pd.DataFrame(data, columns=['x','y'])
X=df[['x']]
y=df['y']

X_train, X_test, y_train, y_test = ms.train_test_split(X, y, test_size=0.4, random_state=50)

model = pipeline.make_pipeline(prep.StandardScaler(), lm.LinearRegression())
model.fit(X_train, y_train)

metrics.r2_score(y_test, model.predict(X_test))

0.8150675457605236
```

Pre-processing

- Standardization
- Non-linear transformation
- Normalization
- Encoding categorical features
- Discretization
- Imputation of missing values
- Generating polynomial features

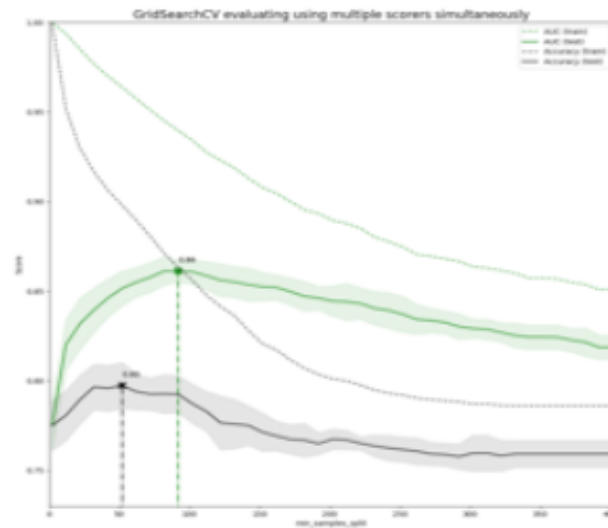
Source: <https://scikit-learn.org/stable/modules/preprocessing.html>

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning

Algorithms: grid search, cross validation, metrics, and more...



Conclusions

- Supervised and Unsupervised learning
- Main characteristics of scikit-learn