# Pythonkurs - 00 - Mandag - Scikit Learn - Intro

December 6, 2024

[1]:
```
import sklearn
print('Version: ', sklearn.__version__)
```

```
Version:  1.5.1
```

[2]:
```
from sklearn import set_config
set_config(display="diagram")
```

## 0.1 Easy start - Training and predictions

[3]:
```
from sklearn.ensemble import RandomForestClassifier

# The classifier
model = RandomForestClassifier(random_state=42)

# Two observations
X = [[1, 2, 3],
    [11, 12, 13]]

# Two possible classes
y = [0, 1]

# Training
model.fit(X, y)
```

[3]:
```
RandomForestClassifier(random_state=42)
```

[4]:
```
# Predict
model.predict(X)
```

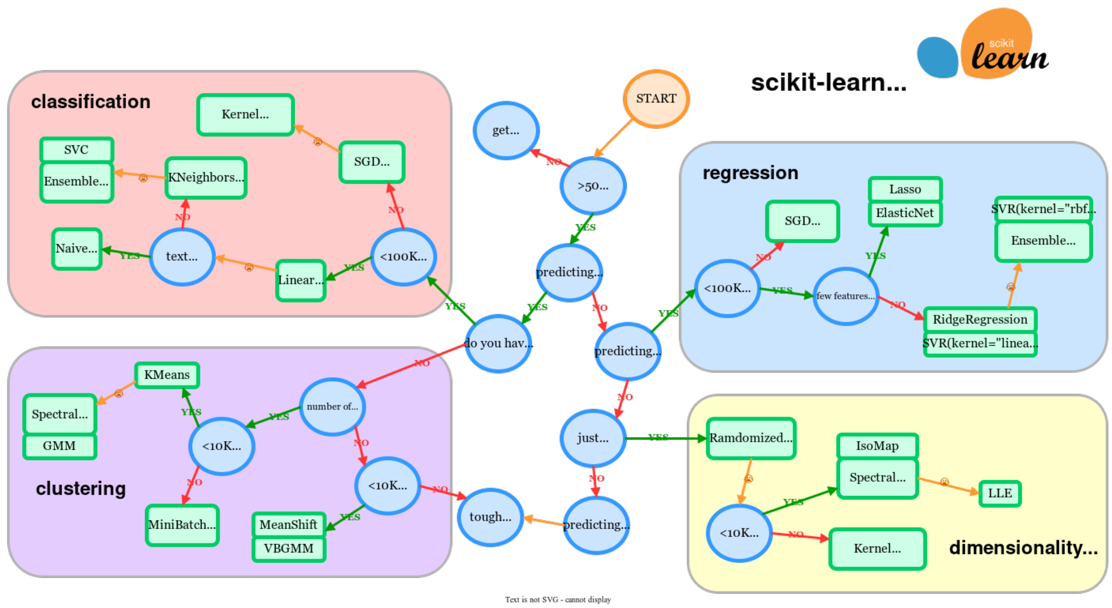
[4]:
```
array([0, 1])
```

[5]:
```
# Predict with new unseen data
model.predict([[14, 15, 16],[4, 5, 6] ])
```

[5]:
```
array([1, 0])
```

[6]:
```
# Predict and show the probabilities
model.predict_proba([[14, 15, 16],[4, 5, 6] ])
```

```
[6]: array([[0.26, 0.74],
            [0.78, 0.22]])
```




https://scikit-learn.org/stable/machine_learning_map.html

| Supervised learning | Unsupervised learning |
|---|---|
| modules/linear_model | modules/mixture |
| modules/lda_qda | modules/manifold |
| modules/kernel_ridge | modules/clustering |
| modules/svm | modules/clustering |
| modules/sgd | modules/biclustering |
| modules/neighbors | modules/decomposition |
| modules/gaussian_process | modules/covariance |
| modules/cross_decomposition | modules/outlier_detection |
| modules/naive_bayes | modules/density |
| modules/tree | modules/neural_networks_unsupervised |
| modules/ensemble | |
| modules/multiclass | |
| modules/feature_selection | |
| modules/semi_supervised | |
| modules/isotonic | |
| modules/calibration | |
| modules/neural_networks_supervised | |

https://scikit-learn.org/stable/user_guide.html

## 0.2 Another example

[7]:
```python
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

X, y = load_iris(return_X_y=True)
```

[8]:
```python
# Inspect the data before proceeding
print(type(X)) # Type of variable X
print(type(y)) # Type of variable y

print(len(X)) # Length of the ndarray X
print(len(y)) # Length of the ndarray y

print(X[:5]) # First n elements of ndarray X
print(y[:5]) # First n elements of ndarray y
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
150
150
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
[0 0 0 0 0]
```

[9]:
```python
clf = LogisticRegression(random_state=42,max_iter=1000).fit(X, y)
```

[10]:
```python
clf.predict(X[:5, :])
```

[10]:
```
array([0, 0, 0, 0, 0])
```

[11]:
```python
clf.predict_proba(X[:5, :])
```

[11]:
```
array([[9.81553399e-01, 1.84465869e-02, 1.45568873e-08],
       [9.71275279e-01, 2.87246911e-02, 3.03078309e-08],
       [9.85243638e-01, 1.47563500e-02, 1.23933441e-08],
       [9.76019242e-01, 2.39807177e-02, 3.98628067e-08],
       [9.85211380e-01, 1.47886080e-02, 1.20542926e-08]])
```

[12]:
```python
clf.score(X, y)
```

[12]:
```
0.9733333333333334
```

## 0.3 Second gear - Transformers and pre-processors

[13]:
```
from sklearn.preprocessing import StandardScaler
import numpy as np

x = np.array([1,2,3,4,5,6])
print(x) # Show the effect of the Numpy reshape, x is a row
X = np.array([1,2,3,4,5,6]).reshape(-1, 1)
print(X) # Show the effect of the Numpy reshape, X is a column

# Create a pre-processor, i.e StandardScaler = Standardize features by removing␣
↪the mean and scaling to unit variance.
scaler = StandardScaler().fit(X)
scaler.transform(X)
```

```
[1 2 3 4 5 6]
[[1]
 [2]
 [3]
 [4]
 [5]
 [6]]
```

[13]:
```
array([[-1.46385011],
       [-0.87831007],
       [-0.29277002],
       [ 0.29277002],
       [ 0.87831007],
       [ 1.46385011]])
```

[14]:
```
# Chain it if that suits you better. Same result.
StandardScaler().fit(X).transform(X)
```

[14]:
```
array([[-1.46385011],
       [-0.87831007],
       [-0.29277002],
       [ 0.29277002],
       [ 0.87831007],
       [ 1.46385011]])
```

[15]:
```
# All paths leading to the same place.
StandardScaler().fit_transform(X)
```

[15]:
```
array([[-1.46385011],
       [-0.87831007],
       [-0.29277002],
       [ 0.29277002],
       [ 0.87831007],
```

```
        [ 1.46385011]])
```

[16]:
```
XX = np.array([1.1,2.2,3.3,4.4,5.5,6.6]).reshape(-1, 1)
scaler.transform(XX) # Reuse the calculated values to transform another dataset.
```

[16]:
```
array([[-1.40529611],
       [-0.76120206],
       [-0.11710801],
       [ 0.52698604],
       [ 1.17108009],
       [ 1.81517414]])
```

[17]:
```
import pandas as pd

X = pd.DataFrame(
    {'city': ['Oslo', 'Bergen', 'Tromsø', 'Oslo'],
     'slogan': ["Unanimiter et constanter", "Byen mellom de syv fjell",
     "Nordens Paris", "Tenk om"],
     'tripadvisor_rating': [5, 3, 4, 5],
     'hotels_rating': [4, 5, 4, 3]})
```

[18]:
```
X
```

[18]:

| | city | slogan | tripadvisor_rating | hotels_rating |
|---|---|---|---|---|
| 0 | Oslo | Unanimiter et constanter | 5 | 4 |
| 1 | Bergen | Byen mellom de syv fjell | 3 | 5 |
| 2 | Tromsø | Nordens Paris | 4 | 4 |
| 3 | Oslo | Tenk om | 5 | 3 |

[19]:
```
from sklearn.compose import ColumnTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import OneHotEncoder
column_trans = ColumnTransformer(
    [('encode_cities', OneHotEncoder(dtype='int'), ['city']),
    ('vectorize_slogan', CountVectorizer(), 'slogan')],
    remainder='drop', verbose_feature_names_out=False)
```

[20]:
```
column_trans.fit(X)
```

[20]:
```
ColumnTransformer(transformers=[('encode_cities', OneHotEncoder(dtype='int'),
                                 ['city']),
                                ('vectorize_slogan', CountVectorizer(),
                                 'slogan')],
                  verbose_feature_names_out=False)
```

[21]:
```
column_trans.get_feature_names_out()
```

[21]: array(['city_Bergen', 'city_Oslo', 'city_Tromsø', 'byen', 'constanter',
       'de', 'et', 'fjell', 'mellom', 'nordens', 'om', 'paris', 'syv',
       'tenk', 'unanimiter'], dtype=object)

[22]:
```
column_trans.transform(X).toarray()
```

[22]: array([[0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]])

## 0.4 Speeding up - Pipeplines

[23]:
```
from sklearn.preprocessing import StandardScaler , MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

pipe = make_pipeline(
    StandardScaler(), # Pre-processor / Transformer / fit() + transform()
    MinMaxScaler(), # Pre-processor / Transformer / fit() + transform()
    LogisticRegression() # Estimator  / Classifier / fit() + predict()
)
```

[24]:
```
X, y = load_iris(return_X_y=True) # Same as before.
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

[25]:
```
print(type(X_train)) # Type of variable
print(type(X_test)) # Type of variable
print(type(y_train)) # Type of variable
print(type(y_test)) # Type of variable

print(len(X_train)) # Length of the ndarray
print(len(X_test)) # Length of the ndarray
print(len(y_train)) # Length of the ndarray
print(len(y_test)) # Length of the ndarray

print(X_train[:5]) # First n elements of ndarray
print(X_test[:5]) # First n elements of ndarray
print(y_train[:5]) # First n elements of ndarray
print(y_test[:5]) # First n elements of ndarray
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

```
112
38
112
38
[[5.  3.6 1.4 0.2]
 [5.2 4.1 1.5 0.1]
 [5.8 2.7 5.1 1.9]
 [6.  3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]]
[[6.1 2.8 4.7 1.2]
 [5.7 3.8 1.7 0.3]
 [7.7 2.6 6.9 2.3]
 [6.  2.9 4.5 1.5]
 [6.8 2.8 4.8 1.4]]
[0 0 2 1 1]
[1 0 2 1 1]
```

[26]:
```
pipe.fit(X_train, y_train)
```

[26]:
```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('minmaxscaler', MinMaxScaler()),
                ('logisticregression', LogisticRegression())])
```

[27]:
```
pipe.predict(X_test)
```

[27]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 2, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0])
```

[28]:
```
accuracy_score(pipe.predict(X_test), y_test)
```

[28]:
```
0.9736842105263158
```

### 0.5 Get ready for landing - Evaluation

[29]:
```
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate

X, y = make_regression(n_samples=100, random_state=42) # Generata some data,␣
↪100 observations.
lr = LinearRegression() # Estimator
```

[30]:
```
print(type(X)) # Type of the variable
print(type(y)) # Type of the variable

print(len(X)) # Length of the ndarray
print(len(y)) # Length of the ndarray
```

```
print(X[:1]) # First n elements of ndarray
print(y[:1]) # First n elements of ndarray
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
100
100
[[ 0.97141236  1.41484066 -0.1118471  -0.87419927 -0.58773797 -0.45517178
   1.15724455  0.56660215 -0.1714956   0.77982864  0.22525668 -0.45121883
   1.25619744  0.90678727  0.29408204  1.47812123  1.56851899  0.14647561
  -0.06433765 -0.13080059 -1.16466335 -0.69380442  0.88484258 -1.18755317
   1.1835491   0.32018785  1.46171702  1.43383284 -0.2724269   0.18745938
  -1.954593    1.18771338 -0.75134547 -0.13833528  0.65692907  1.52009174
   3.15777128 -1.48239679 -1.20391365 -1.16786531  0.29282884 -0.34333518
  -1.15860326  2.05124694  0.66359787 -1.88043554  0.52332378 -2.1146936
  -0.78804762  1.1077211   1.14220739  0.95593582 -0.49731715  0.638187
   0.57890982 -0.80059034 -1.29525862 -0.09931434  1.01562788 -0.3774234
   1.26925567 -0.30950989 -0.65609724  0.27751586 -0.2332085  -0.3103968
  -0.37550898 -0.23927315  0.19200424  0.62936063 -1.44320079  0.16333468
  -1.37729549 -0.37510899  0.13111932 -0.27583996  0.20479788 -0.29906909
  -0.73467109 -0.45269015 -0.42390107  0.06323215 -1.81564898  0.20713944
   0.88106219 -0.63688239 -0.71013661  1.5920249   0.40914092  0.41314748
  -0.32837526  0.89765577  0.61908317  0.6443105   1.74431111  0.05245739
  -1.5693896   0.54058954 -0.5140888   0.67929382]]
[158.47584755]
```

[31]:
```
result = cross_validate(lr, X, y)
```

[32]:
```
result['test_score']
```

[32]:
```
array([0.75402874, 0.84904676, 0.76586317, 0.79373959, 0.86423437])
```

## 0.6 Land ahoy - Automatic parameter search

[33]:
```
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from scipy.stats import randint

X, y = fetch_california_housing(return_X_y=True) # https://scikit-learn.org/
 ↪stable/datasets/real_world.html#california-housing-dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42) #␣
 ↪Train-test-split as previously seen.
```

```
param_distributions = {'n_estimators': randint(1, 5),
                       'max_depth': randint(5, 10)}

search = RandomizedSearchCV(estimator=RandomForestRegressor(random_state=42),
                            n_iter=5, # Test 5 random combinations of the␣
 ↪params.
                            param_distributions=param_distributions, # The␣
 ↪parameters of interest.
                            random_state=42)
```

[34]:
```
print(type(X_train)) # Type of variable
print(type(X_test)) # Type of variable
print(type(y_train)) # Type of variable
print(type(y_test)) # Type of variable

print(len(X_train)) # Length of the ndarray
print(len(X_test)) # Length of the ndarray
print(len(y_train)) # Length of the ndarray
print(len(y_test)) # Length of the ndarray

print(X_train[:5]) # First n elements of ndarray
print(X_test[:5]) # First n elements of ndarray
print(y_train[:5]) # First n elements of ndarray
print(y_test[:5]) # First n elements of ndarray
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
15480
5160
15480
5160
[[ 4.21430000e+00  3.70000000e+01  5.28823529e+00  9.73529412e-01
   8.60000000e+02  2.52941176e+00  3.38100000e+01 -1.18120000e+02]
 [ 5.34680000e+00  4.20000000e+01  6.36432161e+00  1.08793970e+00
   9.57000000e+02  2.40452261e+00  3.71600000e+01 -1.21980000e+02]
 [ 3.91910000e+00  3.60000000e+01  6.11006289e+00  1.05974843e+00
   7.11000000e+02  2.23584906e+00  3.84500000e+01 -1.22690000e+02]
 [ 6.37030000e+00  3.20000000e+01  6.00000000e+00  9.90196078e-01
   1.15900000e+03  2.27254902e+00  3.41600000e+01 -1.18410000e+02]
 [ 2.36840000e+00  1.70000000e+01  4.79585799e+00  1.03550296e+00
   7.06000000e+02  2.08875740e+00  3.85700000e+01 -1.21330000e+02]]
[[ 1.68120000e+00  2.50000000e+01  4.19220056e+00  1.02228412e+00
   1.39200000e+03  3.87743733e+00  3.60600000e+01 -1.19010000e+02]
 [ 2.53130000e+00  3.00000000e+01  5.03938356e+00  1.19349315e+00
   1.56500000e+03  2.67979452e+00  3.51400000e+01 -1.19460000e+02]
 [ 3.48010000e+00  5.20000000e+01  3.97715472e+00  1.18587747e+00
```

```
  1.31000000e+03  1.36033229e+00  3.78000000e+01 -1.22440000e+02]
 [ 5.73760000e+00  1.70000000e+01  6.16363636e+00  1.02020202e+00
   1.70500000e+03  3.44444444e+00  3.42800000e+01 -1.18720000e+02]
 [ 3.72500000e+00  3.40000000e+01  5.49299065e+00  1.02803738e+00
   1.06300000e+03  2.48364486e+00  3.66200000e+01 -1.21930000e+02]]
[2.285 2.799 1.83  4.658 1.5  ]
[0.477   0.458   5.00001 2.186   2.78   ]
```

[35]:
```
search.fit(X_train, y_train)
```

[35]:
```
RandomizedSearchCV(estimator=RandomForestRegressor(random_state=42), n_iter=5,
                   param_distributions={'max_depth':
<scipy.stats._distn_infrastructure.rv_discrete_frozen object at 0x17fee7500>,
                                        'n_estimators':
<scipy.stats._distn_infrastructure.rv_discrete_frozen object at 0x17e8e08c0>},
                   random_state=42)
```

[36]:
```
search.best_params_
```

[36]:
```
{'max_depth': 7, 'n_estimators': 4}
```

[37]:
```
results = pd.DataFrame(search.cv_results_)[['params', 'mean_test_score',␣
↪'rank_test_score']]
results = results.sort_values('rank_test_score')
results
```

[37]:

| | params | mean_test_score | rank_test_score |
|---|---|---|---|
| 1 | {'max_depth': 7, 'n_estimators': 4} | 0.707346 | 1 |
| 4 | {'max_depth': 7, 'n_estimators': 3} | 0.698350 | 2 |
| 3 | {'max_depth': 6, 'n_estimators': 3} | 0.674247 | 3 |
| 0 | {'max_depth': 8, 'n_estimators': 1} | 0.657089 | 4 |
| 2 | {'max_depth': 9, 'n_estimators': 1} | 0.654602 | 5 |

[38]:
```
search.score(X_test, y_test) # Check score on our testdata.
```

[38]:
```
0.7094093293239057
```

[39]:
```
search.best_estimator_.score(X_test, y_test) # Another way to check score on␣
↪our testdata.
```

[39]:
```
0.7094093293239057
```