Scikit-learn

January 16, 2021

1 Uczenie maszynowe w Pythonie

1.0.1 Ekosystem

- pandas
- numpy
- scikit-learn
- matplotlib, seaborns, bokeh, pyplot

1.0.2 Co jeszcze:

- pytorch
- tensorflow (keras)

1.1 Numpy

- biblioteka, która jest podstawą pandasa, scikit-learn i pozostałych bibliotek.
- Wprowadza nowy typ danych tablice wielowymiarowe: `np.array`.
- Ponadto dostarcza wydajne funkcje, które operują na `np.array`

[203]:
```
import numpy as np

a = np.array([1, 2, 3])
print(a)
```

```
[1 2 3]
```

[208]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
a
```

[208]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

[213]:
```
a[0][1:3]
```

[213]:
```
array([2, 3])
```

[200]:
```
np.log(a)
```

[200]:
```
array([[0.        , 0.69314718, 1.09861229],
       [1.38629436, 1.60943791, 1.79175947]])
```

[201]:
```
a.shape
```

[201]:
```
(2, 3)
```

[217]:
```
a[0][0] = '10'
a
```

[217]:
```
array([[10,  2,  3],
       [ 4,  5,  6]])
```

https://numpy.org/doc/1.19/user/tutorials_index.html

1.1.1 Scikit-learn

- prosta obsługa
- kompaktybilne z innymi bibliotekami
- Darmowe, open-source (BSD)

1.1.2 Scikit-learn

- przetwadzanie danych
- modele
- trening i ewaluacja

[218]:
```
import sklearn
import pandas as pd
```

[219]:
```
from sklearn import datasets
```

[224]:
```
data = datasets.load_iris()

from sklearn.model_selection import train_test_split

features = pd.DataFrame(data['data'], columns=data['feature_names'])
labels = pd.Series(data['target'])

features.head()
labels.sample(10)
```

[224]:
```
97     1
29     0
123    2
98     1
83     1
125    2
```

```
14     0
85     1
74     1
1      0
dtype: int64
```

[229]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

train_x, test_x, train_y, test_y = train_test_split(features, labels,␣
 ↪test_size=0.33, random_state=0)

model = KNeighborsClassifier(5)
model.fit(train_x, train_y)

predictions = model.predict(test_x)
accuracy_score(predictions, test_y)

print(predictions)
print(test_y.values)
```

```
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2 1 1 2 0 2 0 0 1 2 2 2 2]
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1 1 1 2 0 2 0 0 1 2 2 2 2]
```

Zadanie 1: znajdz najlepsze k w algorytmie KNN

1.2 Przetwarzanie danych

[38]:
```
from sklearn import preprocessing
```

1.2.1 Co może wchodzić w skład przetwarzania

- skalowanie (`StandardScaler`, `MinMaxScaler`, `MaxAbsScaler`, `Normalizer`)
- redukcja wymiaru (`PCA`)
- enkodowanie (`LabelEncoder`)

- Bardzo żadko zdarza się, żeby dane były gotowe do treningu.
- Najczęściej musimy je przetworzyć, aby móc je wykorzystać i/lub uzyskać lepszy wynik.
- scikit-learn potrafi operować wyłącznie na danych numerycznych.

1.2.2 Transformery:

- służą do przetwarzania danych
- mają metody: `fit`, `transform` i `fit_transform`.

[42]:
```
features.head()
```

[42]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

[53]:
```
scaler = preprocessing.StandardScaler()
scaler.fit(train_x)
train_scaled = scaler.transform(train_x)

train_scaled[:5]
```

[53]:
```
array([[-0.31699245, -1.33490866,  0.07470002, -0.13206311],
       [ 2.06897222, -0.15774053,  1.2920336 ,  1.40653626],
       [ 0.47832911,  0.54856035,  0.51736677,  0.50901996],
       [-0.43060981, -1.57034229,  0.01936667, -0.13206311],
       [ 0.47832911, -0.86404141,  0.62803346,  0.76545319]])
```

[62]:
```
print('Mean:', scaler.mean_)
print()
print('Scale:', scaler.scale_)
```

```
Mean: [5.879 3.067 3.765 1.203]

Scale: [0.88014715 0.42474816 1.80722854 0.77993013]
```

[61]:
```
model = KNeighborsClassifier(5)
model.fit(train_scaled, train_y)

predictions = model.predict(scaler.transform(test_x))
accuracy_score(predictions, test_y)
```

[61]:
```
0.96
```

Zadanie 2

[64]:
```
from sklearn.datasets import load_wine
wine = load_wine()
train_x, test_x, train_y, test_y = train_test_split(wine.data, wine.target,␣
 ↪test_size=0.2, random_state=0)
model = KNeighborsClassifier(5)
model.fit(train_x, train_y)

predictions = model.predict(test_x)
accuracy_score(predictions, test_y)
```

[80]:
```
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

wine = load_wine()
train_x, test_x, train_y, test_y = train_test_split(wine.data, wine.target,␣
↪test_size=0.2, random_state=0)

scaler = StandardScaler()
pca = PCA(2)


model = KNeighborsClassifier(5)
model.fit(train_x, train_y)

predictions = model.predict(test_x)
accuracy_score(predictions, test_y)
```

[80]:
```
0.8055555555555556
```

[82]:
```
scaled_train_x = scaler.fit_transform(train_x)
decomposed_train_x = pca.fit_transform(scaled_train_x)

model = KNeighborsClassifier(5)
model.fit(decomposed_train_x, train_y)

predictions = model.predict(
    pca.transform(
        scaler.transform(test_x)
    )
)
accuracy_score(predictions, test_y)
```

[82]:
```
0.9722222222222222
```

1.2.3 Pipeline

[82]:
```
scaled_train_x = scaler.fit_transform(train_x)
decomposed_train_x = pca.fit_transform(scaled_train_x)

model = KNeighborsClassifier(5)
model.fit(decomposed_train_x, train_y)

predictions = model.predict(
    pca.transform(
        scaler.transform(test_x)
    )
```

```
)
accuracy_score(predictions, test_y)
```

[82]: 0.9722222222222222

[89]:
```
from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(StandardScaler(), PCA(2), KNeighborsClassifier(5))
pipeline.fit(train_x, train_y)

predictions = pipeline.predict(test_x)
accuracy_score(predictions, test_y)
```

[89]: 0.9722222222222222

[189]:
```
pipeline.steps
```

[189]:
```
[('columntransformer-1',
  ColumnTransformer(remainder='passthrough',
                    transformers=[('encoder', LabelEncoder(), 'Sex')])),
 ('columntransformer-2',
  ColumnTransformer(remainder='passthrough',
                    transformers=[('encoder', OneHotEncoder(), 'Embarked')]))]
```

[190]:
```
pipeline.steps[1]
```

[190]:
```
('columntransformer-2',
 ColumnTransformer(remainder='passthrough',
                   transformers=[('encoder', OneHotEncoder(), 'Embarked')]))
```

1.2.4 Obsługa danych nienumerycznych

- `OrdinalEncoder` lub `LabelEncoder`
- `OneHotEncoder`

[95]:
```
titanic = pd.read_csv('/home/tomaszd/codes/Python2021/labs02/titanic_train.
↪tsv', delimiter='\t')
```

[97]:
```
titanic.head()
```

[97]:

	PassengerId	Survived	Pclass	\
0	1	0	3	
1	2	1	1	
2	3	1	3	
3	4	1	1	
4	5	0	3	

```
                                                 Name     Sex   Age  SibSp  \
0                            Braund\t Mr. Owen Harris    male  22.0      1
1  Cumings\t Mrs. John Bradley (Florence Briggs T…  female  38.0      1
2                             Heikkinen\t Miss. Laina  female  26.0      0
3       Futrelle\t Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen\t Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[160]:
```
from sklearn.preprocessing import OrdinalEncoder

encoder = OrdinalEncoder()
encoder.fit(titanic[['Sex']])

print('klasy:', encoder.categories_)
print(encoder.transform(titanic[['Sex']])[:10])
```

```
klasy: [array(['female', 'male'], dtype=object)]
[[1.]
 [0.]
 [0.]
 [0.]
 [1.]
 [1.]
 [1.]
 [1.]
 [0.]
 [0.]]
```

[128]:
```
from sklearn.preprocessing import OneHotEncoder

print(titanic[['Embarked']].loc[:3])

encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
encoder.fit(titanic[['Embarked']].dropna())
print(encoder.categories_)

print(encoder.transform(titanic[['Embarked']])[:3])
```

```
  Embarked
0        S
1        C
2        S
```

```
3        S
[array(['C', 'Q', 'S'], dtype=object)]
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

1.3 Łączenie transformerów

- Transformery przetwarzają wszystkie kolumny jednocześnie.
- Jeżeli chcemy zastować dany transformer dla jednej cechy (kolumny), możemy użyć `ColumnTransformer`.

[149]:
```
from sklearn.compose import ColumnTransformer

col_1 = ColumnTransformer(
    [
        ('encoder_sex', OneHotEncoder(handle_unknown='ignore', sparse=False),␣
 ↪['Sex']),
        ('encoder_emb', OneHotEncoder(handle_unknown='ignore', sparse=False),␣
 ↪['Embarked'])
    ],
    remainder='passthrough')

col_1.fit_transform(titanic)[:2]
```

[149]:
```
array([[0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1, 0, 3,
        'Braund\t Mr. Owen Harris', 22.0, 1, 0, 'A/5 21171', 7.25, nan],
       [1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 2, 1, 1,
        'Cumings\t Mrs. John Bradley (Florence Briggs Thayer)', 38.0, 1,
        0, 'PC 17599', 71.2833, 'C85']], dtype=object)
```

zadanie 3

[187]:
```
df = pd.read_csv('https://raw.githubusercontent.com/mwaskom/seaborn-data/master/
 ↪tips.csv')
features = df.drop('tip', axis='columns')
features

col = ColumnTransformer(
    [
        ('encoder_1', OrdinalEncoder(handle_unknown='ignore'), ['smoker',␣
 ↪'sex']),
        ('encoder_2', OneHotEncoder(handle_unknown='ignore', sparse=False),␣
 ↪['day', 'time'])
    ],
    remainder='passthrough')

col.fit_transform(features[:1])
```

[187]:
```
array([[ 0.  ,  0.  ,  1.  ,  1.  , 16.99,  2.  ]])
```

2 Modele

- scikit-learn implementuje wiele podstawowych model do uczenia maszynowego:
  - modele regresyjne
  - k-najbliższych sąsiadów
  - SVM
  - i wiele innych.
- mają one wspólny interfejs: `fit`, `predict`, `fit_predict`.
- Wspólny interfejs pozwala na proste przełączanie się między modelami.

[169]:
```
iris = datasets.load_iris()
train_x, test_x, train_y, test_y = train_test_split(iris.data, iris.target,␣
↪test_size=0.2, random_state=0)

model = KNeighborsClassifier(5)
model.fit(train_x, train_y)
model.predict(test_x)
```

[169]:
```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 2, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0])
```

[171]:
```
from sklearn.svm import SVC

models = [KNeighborsClassifier(5), SVC()]
for model in models:
    model.fit(train_x, train_y)
    print(model.score(test_x, test_y))
```

```
0.9666666666666667
1.0
```

[188]:
```
df = pd.read_csv('/home/tomaszd/codes/Python2021/labs01/gapminder.csv')
len(df)
```

[188]:
```
175
```

[186]:
```
Zadanie 4:

x = df.drop(['Country', 'life_expectancy'], axis='columns')
y = df['life_expectancy']
import numpy as np
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LinearRegression
col = ColumnTransformer(
```

```
    [
        ('encoder_1', FunctionTransformer(np.log), ['population', 'gdp']),
    ],
    remainder='passthrough')

pipe = make_pipeline(col, LinearRegression())
pipe.fit(x,y)

print(pipe.steps[1][1].coef_)
print("Wyraz wolny (bias):", pipe.steps[1][1].intercept_)
```

```
[ 0.21064949  1.6285011  -1.11712805  1.32890756 -0.15096735  0.79475121]
Wyraz wolny (bias): 51.430517627456545
```

2.1 Walidacja krzyżowa

- Czasami nasze dane są zbyt małe, żeby móc wydzielić zbiór testowy.

[194]:
```
from sklearn.model_selection import cross_val_score
iris = datasets.load_iris()

knn = KNeighborsClassifier(n_neighbors=5)

scores = cross_val_score(knn, iris.data, iris.target, cv=10, scoring='accuracy')

print("Wynik walidacji krzyżowej:", scores)
```

```
Wynik walidacji krzyżowej: [1.         0.93333333 1.         1.
0.86666667 0.93333333
 0.93333333 1.         1.         1.        ]
```