W2.code

JUNHYEOK 김준수 ㅁㅁㅁ

2025-03-27 오후 1:12

Jupyter Applied_

```
# Step-by-Step Supervised Machine Learning Modeling with Python

Jose Gonzalez Orehuela
Data Science Consultant, Author, Speaker, Instructor
Joint Center for Housing Studies, Harvard University
Cambridge, MA, USA
```

Step 1 of 4

Step 1 of 4 | Step 2 of 4 | Step 3 of 4 | Step 4 of 4 | Home | Feedback

Step 1 of 4: Learning Supervised Machine Learning with Python - Applied Machine Learning in Python - Coursera

Coursera | Home | Feedback

Step 1 of 4 Instructions

Hide Code | Code | Cell | View | Kernel | Help | Trusted | Jupyter Notebook | Python 3 (ipykernel)

Applied Machine Learning Module 2 (Supervised Learning, Part 1)

Preamble and Review

```
%matplotlib inline
import numpy as np
import scipy as sp
import matplotlib as mpl
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import pandas as pd
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 100)
pd.set_option('display.notebook_repeat_output', True)

import sklearn as sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.datasets import load_digits
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import classification_report
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import scale
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import KFold
from sklearn.grid_search import GridSearchCV
from sklearn.grid_search import RandomizedSearchCV
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

```
%matplotlib inline
import numpy as np
import scipy as sp
import matplotlib as mpl
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import pandas as pd
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 100)
pd.set_option('display.notebook_repeat_output', True)

import sklearn as sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.datasets import load_digits
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_curve, average_precision_score
from sklearn.metrics import classification_report
from sklearn.preprocessing import Imputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import scale
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import KFold
from sklearn.grid_search import GridSearchCV
from sklearn.grid_search import RandomizedSearchCV
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

Exercise

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys(): \n{}".format(cancer.keys()))
print("Shape of cancer.data: {}".format(cancer.data.shape))
print("Sample counts per class:\n{}".format(
    {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))
print("Feature names:\n{}".format(cancer.feature_names))
```

```
cancer.keys(): 
dict_keys(['target_names', 'data', 'target', 'DESCR', 'feature_names'])
Shape of cancer.data: (569, 30)
Sample counts per class:
{'malignant': 212, 'benign': 357}
Feature names:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

```
from sklearn.datasets import load_digits
digits = load_digits()
print("digits.keys(): \n{}".format(digits.keys()))
print("Shape of digits.data: {}".format(digits.data.shape))
print("Sample counts per class:\n{}".format(
    {n: v for n, v in zip(digits.target_names, np.bincount(digits.target))}))
print("Feature names:\n{}".format(digits.feature_names))
```

```
digits.keys(): 
dict_keys(['target_names', 'data', 'target', 'DESCR', 'feature_names', 'images'])
Shape of digits.data: (1797, 64)
Sample counts per class:
{'0': 178, '1': 182, '2': 177, '3': 183, '4': 181, '5': 182, '6': 181, '7': 179, '8': 174, '9': 180}
Feature names:
['pixel_0_0' 'pixel_0_1' 'pixel_0_2' 'pixel_0_3' 'pixel_0_4' 'pixel_0_5'
 'pixel_0_6' 'pixel_0_7' 'pixel_1_0' 'pixel_1_1' 'pixel_1_2' 'pixel_1_3'
 'pixel_1_4' 'pixel_1_5' 'pixel_1_6' 'pixel_1_7' 'pixel_2_0' 'pixel_2_1'
 'pixel_2_2' 'pixel_2_3' 'pixel_2_4' 'pixel_2_5' 'pixel_2_6' 'pixel_2_7'
 'pixel_3_0' 'pixel_3_1' 'pixel_3_2' 'pixel_3_3' 'pixel_3_4' 'pixel_3_5'
 'pixel_3_6' 'pixel_3_7' 'pixel_4_0' 'pixel_4_1' 'pixel_4_2' 'pixel_4_3'
 'pixel_4_4' 'pixel_4_5' 'pixel_4_6' 'pixel_4_7' 'pixel_5_0' 'pixel_5_1'
 'pixel_5_2' 'pixel_5_3' 'pixel_5_4' 'pixel_5_5' 'pixel_5_6' 'pixel_5_7'
 'pixel_6_0' 'pixel_6_1' 'pixel_6_2' 'pixel_6_3' 'pixel_6_4' 'pixel_6_5'
 'pixel_6_6' 'pixel_6_7' 'pixel_7_0' 'pixel_7_1' 'pixel_7_2' 'pixel_7_3'
 'pixel_7_4' 'pixel_7_5' 'pixel_7_6' 'pixel_7_7']
```

K-Nearest Neighbors

Classification

```
from mglearn.datasets import make_forge
X, y = make_forge()
print("X.shape: {}".format(X.shape))
print("y.shape: {}".format(y.shape))
```

```
X.shape: (26, 2)
y.shape: (26,)
```

Regression

```
from mglearn.datasets import make_wave
X, y = make_wave(n_samples=40)
print("X.shape: {}".format(X.shape))
print("y.shape: {}".format(y.shape))
```

```
X.shape: (40, 1)
y.shape: (40,)
```

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys(): \n{}".format(cancer.keys()))
print("Shape of cancer.data: {}".format(cancer.data.shape))
print("Sample counts per class:\n{}".format(
    {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))
print("Feature names:\n{}".format(cancer.feature_names))
```

```
cancer.keys(): 
dict_keys(['target_names', 'data', 'target', 'DESCR', 'feature_names'])
Shape of cancer.data: (569, 30)
Sample counts per class:
{'malignant': 212, 'benign': 357}
Feature names:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

Regression model complexity as a function of K

```
from mglearn.datasets import load_boston
boston = load_boston()
print("Data shape: {}".format(boston.data.shape))
```

```
Data shape: (506, 13)
```

Linear models for regression

Linear Regression

```
from mglearn.datasets import load_extended_boston
X, y = mglearn.datasets.load_extended_boston()
print("X.shape: {}".format(X.shape))
```

```
X.shape: (506, 104)
```

Linear regression, example plot

```
from sklearn.linear_model import LinearRegression
X, y = mglearn.datasets.make_wave(n_samples=60)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
lr = LinearRegression().fit(X_train, y_train)
print("lr.coef_: {}".format(lr.coef_))
print("lr.intercept_: {}".format(lr.intercept_))
```

```
lr.coef_: [ 0.394]
lr.intercept_: -0.031804343026759685
```

```
print("lr.score(X_train): {:.2f}".format(lr.score(X_train, y_train)))
print("lr.score(X_test): {:.2f}".format(lr.score(X_test, y_test)))
```

```
lr.score(X_train): 0.67
lr.score(X_test): 0.66
```

Ridge regression

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)
print("lr.score(X_train): {:.2f}".format(lr.score(X_train, y_train)))
print("lr.score(X_test): {:.2f}".format(lr.score(X_test, y_test)))
```

```
lr.score(X_train): 0.95
lr.score(X_test): 0.607
```

```
from sklearn.linear_model import Ridge
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
ridge = Ridge().fit(X_train, y_train)
print("ridge.score(X_train): {:.2f}".format(ridge.score(X_train, y_train)))
print("ridge.score(X_test): {:.2f}".format(ridge.score(X_test, y_test)))
```

```
ridge.score(X_train): 0.89
ridge.score(X_test): 0.752
```

Ridge regression model complexity as a function of α

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
for alpha in [10, 1, 0.1, 0.01, 0.001, 0.0001, 0.00001]:
    ridge = Ridge(alpha=alpha).fit(X_train, y_train)
    print("alpha={:.2f}".format(alpha))
    print("ridge.score(X_train): {:.2f}".format(ridge.score(X_train, y_train)))
    print("ridge.score(X_test): {:.2f}".format(ridge.score(X_test, y_test)))
    print("ridge.coef_.shape: {}".format(ridge.coef_.shape))
    print("ridge.coef_.min(): {:.2f}".format(ridge.coef_.min()))
    print("ridge.coef_.max(): {:.2f}".format(ridge.coef_.max()))
    print("ridge.coef_.mean(): {:.2f}".format(ridge.coef_.mean()))
```

```
alpha=10.00
ridge.score(X_train): 0.89
ridge.score(X_test): 0.752
ridge.coef_.shape: (104,)
ridge.coef_.min(): -3.08
ridge.coef_.max(): 2.44
ridge.coef_.mean(): 0.039
alpha=1.00
ridge.score(X_train): 0.95
ridge.score(X_test): 0.607
ridge.coef_.shape: (104,)
ridge.coef_.min(): -4.28
ridge.coef_.max(): 3.37
ridge.coef_.mean(): 0.023
alpha=0.10
ridge.score(X_train): 0.95
ridge.score(X_test): 0.607
ridge.coef_.shape: (104,)
ridge.coef_.min(): -4.64
ridge.coef_.max(): 3.57
ridge.coef_.mean(): 0.022
alpha=0.01
ridge.score(X_train): 0.95
ridge.score(X_test): 0.607
ridge.coef_.shape: (104,)
ridge.coef_.min(): -4.78
ridge.coef_.max(): 3.70
ridge.coef_.mean(): 0.022
alpha=0.001
ridge.score(X_train): 0.95
ridge.score(X_test): 0.607
ridge.coef_.shape: (104,)
ridge.coef_.min(): -4.80
ridge.coef_.max(): 3.73
ridge.coef_.mean(): 0.022
alpha=0.0001
ridge.score(X_train): 0.95
ridge.score(X_test): 0.607
ridge.coef_.shape: (104,)
ridge.coef_.min(): -4.81
ridge.coef_.max(): 3.74
ridge.coef_.mean(): 0.022
alpha=0.00001
ridge.score(X_train): 0.95
ridge.score(X_test): 0.607
ridge.coef_.shape: (104,)
ridge.coef_.min(): -4.81
ridge.coef_.max(): 3.75
ridge.coef_.mean(): 0.022
```

Lasso

```
from sklearn.linear_model import Lasso
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lasso = Lasso().fit(X_train, y_train)
print("lasso.score(X_train): {:.2f}".format(lasso.score(X_train, y_train)))
print("lasso.score(X_test): {:.2f}".format(lasso.score(X_test, y_test)))
print("lasso.coef_.shape: {}".format(lasso.coef_.shape))
print("lasso.coef_.min(): {:.2f}".format(lasso.coef_.min()))
print("lasso.coef_.max(): {:.2f}".format(lasso.coef_.max()))
print("lasso.coef_.mean(): {:.2f}".format(lasso.coef_.mean()))
```

```
lasso.score(X_train): 0.29
lasso.score(X_test): 0.209
lasso.coef_.shape: (104,)
lasso.coef_.min(): -0.00
lasso.coef_.max(): 0.00
lasso.coef_.mean(): 0.000
```

Lasso regression model complexity

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
for alpha in [1, 0.1, 0.01, 0.001, 0.0001, 0.00001]:
    lasso = Lasso(alpha=alpha).fit(X_train, y_train)
    print("alpha={:.2f}".format(alpha))
    print("lasso.score(X_train): {:.2f}".format(lasso.score(X_train, y_train)))
    print("lasso.score(X_test): {:.2f}".format(lasso.score(X_test, y_test)))
    print("lasso.coef_.shape: {}".format(lasso.coef_.shape))
    print("lasso.coef_.min(): {:.2f}".format(lasso.coef_.min()))
    print("lasso.coef_.max(): {:.2f}".format(lasso.coef_.max()))
    print("lasso.coef_.mean(): {:.2f}".format(lasso.coef_.mean()))
```

```
alpha=1.00
lasso.score(X_train): 0.29
lasso.score(X_test): 0.209
lasso.coef_.shape: (104,)
lasso.coef_.min(): -0.00
lasso.coef_.max(): 0.00
lasso.coef_.mean(): 0.000
alpha=0.10
lasso.score(X_train): 0.77
lasso.score(X_test): 0.640
lasso.coef_.shape: (104,)
lasso.coef_.min(): -0.00
lasso.coef_.max(): 4.11
lasso.coef_.mean(): 0.039
alpha=0.01
lasso.score(X_train): 0.95
lasso.score(X_test): 0.607
lasso.coef_.shape: (104,)
lasso.coef_.min(): -4.28
lasso.coef_.max(): 3.37
lasso.coef_.mean(): 0.023
alpha=0.001
lasso.score(X_train): 0.95
lasso.score(X_test): 0.607
lasso.coef_.shape: (104,)
lasso.coef_.min(): -4.64
lasso.coef_.max(): 3.57
lasso.coef_.mean(): 0.022
alpha=0.0001
lasso.score(X_train): 0.95
lasso.score(X_test): 0.607
lasso.coef_.shape: (104,)
lasso.coef_.min(): -4.78
lasso.coef_.max(): 3.70
lasso.coef_.mean(): 0.022
alpha=0.00001
lasso.score(X_train): 0.95
lasso.score(X_test): 0.607
lasso.coef_.shape: (104,)
lasso.coef_.min(): -4.80
lasso.coef_.max(): 3.73
lasso.coef_.mean(): 0.022
```

Linear models for classification

Logistic regression

```
from sklearn.linear_model import LogisticRegression
X, y = mglearn.datasets.make_forge()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
clf = LogisticRegression().fit(X_train, y_train)
print("clf.coef_: {}".format(clf.coef_))
print("clf.intercept_: {}".format(clf.intercept_))
```

```
clf.coef_: [[ 0.447 -0.567]]
clf.intercept_: [-0.037]
```

```
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.60
clf.score(X_test): 0.54
```

Logistic regression on breast cancer dataset

```
from sklearn.linear_model import LogisticRegression
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LogisticRegression().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```

Linear support vector machines

```
from sklearn.linear_model import LinearSVC
X, y = mglearn.datasets.make_forge()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
clf = LinearSVC().fit(X_train, y_train)
print("clf.coef_: {}".format(clf.coef_))
print("clf.intercept_: {}".format(clf.intercept_))
```

```
clf.coef_: [[ 0.220 -0.549]]
clf.intercept_: [ 0.262]
```

```
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.58
clf.score(X_test): 0.54
```

Linear support vector machines: Classification

```
from sklearn.linear_model import LinearSVC
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LinearSVC().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```

Multiclass logistic regression

```
X, y = mglearn.datasets.make_blobs(random_state=42)
mglearn.plotting.scatter_group_3_separate_blobs(X, y)
```

```
X, y = mglearn.datasets.make_blobs(random_state=42)
mglearn.plotting.scatter_group_3_separate_blobs(X, y)
```

Multiclass Support Vector Machines

```
X, y = mglearn.datasets.make_blobs(random_state=42)
mglearn.plotting.scatter_group_3_separate_blobs(X, y)
```

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LogisticRegression().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```

Decision trees

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LinearSVC().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```

Decision trees on breast cancer

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LinearSVC().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```

Decision Trees

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LinearSVC().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LinearSVC().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```

Decision Trees on a moons dataset

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LinearSVC().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```

```
X, y = mglearn.datasets.load_extended_boston()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
clf = LinearSVC().fit(X_train, y_train)
print("clf.score(X_train): {:.2f}".format(clf.score(X_train, y_train)))
print("clf.score(X_test): {:.2f}".format(clf.score(X_test, y_test)))
```

```
clf.score(X_train): 0.95
clf.score(X_test): 0.607
```