Lab 4 / 201835534 차원우

[4-1]

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

# read dataset

ds = pd.read\_csv('housing.csv')

# drop feature 'ocean\_proximity' that has categorical value

ds = ds.drop(['ocean\_proximity'], axis=1)

# make two arrays for finding correlation

income = np.array(ds['median\_income'])

value = np.array(ds['median\_house\_value'])

# split the dataset into training and testing set (4/5, 1/5)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(income, value,\

    test\_size=0.2, random\_state=0, shuffle=True)

# instance fot linear regression, train the model

reg = LinearRegression()

reg.fit(x\_train[:, np.newaxis], y\_train)

# predict using test dataset, show plot

px = x\_test

py = reg.predict(x\_test[:, np.newaxis])

plt.scatter(x\_test, y\_test)

plt.plot(px, py, color='r')

plt.show()

# use score method to get accuracy of the model (evaluation)

score = reg.score(x\_test.reshape(-1,1), y\_test)

print("#1) Evaluate the Regression model: ", score)

# split the dataset into training and testing set (3/5, 2/5)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(income, value,\

    test\_size=0.4, random\_state=0, shuffle=True)

# instance fot linear regression, train the model

reg = LinearRegression()

reg.fit(x\_train[:, np.newaxis], y\_train)

# predict using test dataset, show plot

px = x\_test

py = reg.predict(x\_test[:, np.newaxis])

plt.scatter(x\_test, y\_test)

plt.plot(px, py, color='r')

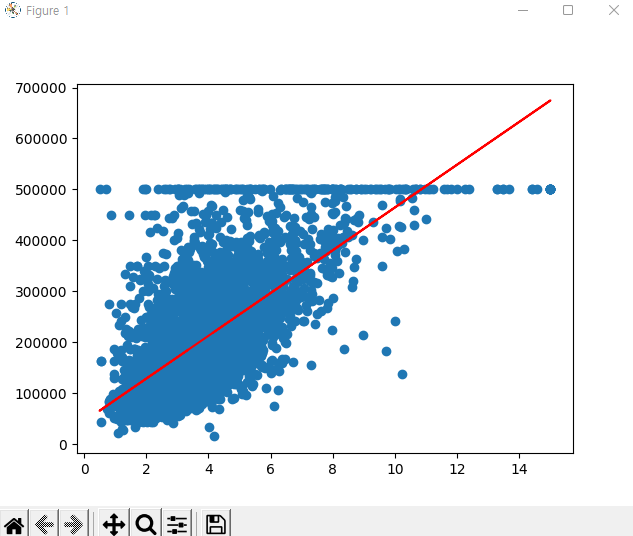
plt.show()

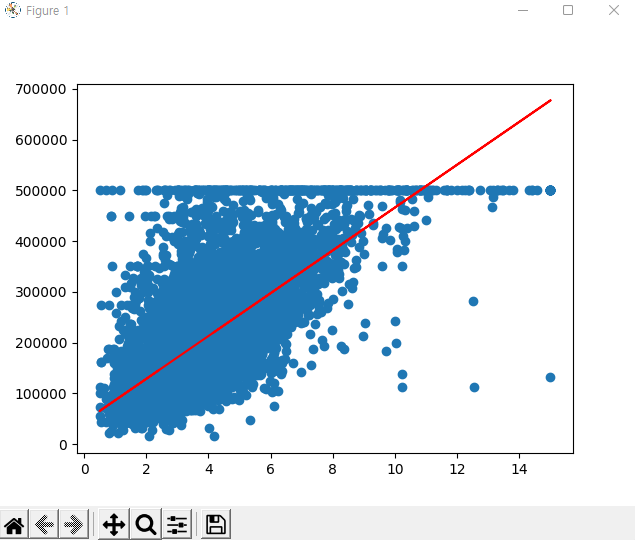
# use score method to get accuracy of the model (evaluation)

score = reg.score(x\_test.reshape(-1,1), y\_test)

print("#2) Evaluate the Regression model: ", score)

- output -







[4-2]

import numpy as np

import pandas as pd

from sklearn import tree

from sklearn.model\_selection import train\_test\_split

# read dataset

ds = pd.read\_csv('winequality-red.csv')

# make two arrays for target feature

others = np.array(ds.iloc[:, 0:11])

quality = np.array(ds['quality'])

# split the dataset into training and testing set (9/10, 1/10)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(others, quality,\

    test\_size=0.1, random\_state=0, shuffle=True)

# construct a tree

clf = tree.DecisionTreeClassifier()

clf = clf.fit(x\_train, y\_train)

##print(tree.plot\_tree(clf))

# predict using test dataset

predictions = clf.predict(x\_test)

# use score method to get accuracy of the model (evaluation)

score = clf.score(x\_test, y\_test)

print("#1) Evaluate the Decision Tree model: ", score)

# split the dataset into training and testing set (8/10, 2/10)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(others, quality,\

    test\_size=0.2, random\_state=0, shuffle=True)

# construct a tree

clf = tree.DecisionTreeClassifier()

clf = clf.fit(x\_train, y\_train)

##print(tree.plot\_tree(clf))

# predict using test dataset

predictions = clf.predict(x\_test)

# use score method to get accuracy of the model (evaluation)

score = clf.score(x\_test, y\_test)

print("#2) Evaluate the Decision Tree model: ", score)

# split the dataset into training and testing set (7/10, 3/10)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(others, quality,\

    test\_size=0.3, random\_state=0, shuffle=True)

# construct a tree

clf = tree.DecisionTreeClassifier()

clf = clf.fit(x\_train, y\_train)

##print(tree.plot\_tree(clf))

# predict using test dataset

predictions = clf.predict(x\_test)

# use score method to get accuracy of the model (evaluation)

score = clf.score(x\_test, y\_test)

print("#3) Evaluate the Decision Tree model: ", score)

- output -

텍스트이(가) 표시된 사진

자동 생성된 설명

[4-3]

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import RandomizedSearchCV

# read dataset

ds = pd.read\_csv('mnist\_test.csv')

# make two arrays for target feature

others = np.array(ds.drop(['label'], axis=1))

quality = np.array(ds['label'])

# prepare cross validation (5-fold)

kfold = KFold(n\_splits=5, shuffle=True, random\_state=1)

# enumerate splits (for k=3)

for train, test in kfold.split(others):

    X\_train = others[train]

    y\_train = quality[train]

    X\_test = others[test]

    y\_test = quality[test]

    # build a KNN model, create KNN classifier (k=3)

    knn\_cv = KNeighborsClassifier(n\_neighbors=3)

    # check accuracy of our model on the test data

    cv\_scores = cross\_val\_score(knn\_cv, X\_test, y\_test, cv=5)

    print("#1) Evaluate the K-NN(k=3) model: ", cv\_scores)

    print("\n#1\_1)Hyper Parameter tuning by GridSearch")

    # create a new KNN model

    knn2 = KNeighborsClassifier()

    # create a dictionary of all values we want to test for n\_neighbors

    param\_grid = {'n\_neighbors': np.arange(1,25)}

    # use GridSearch to test all values for n\_neighbors

    knn\_gscv = GridSearchCV(knn2, param\_grid, cv=5)

    # fit model to data

    knn\_gscv.fit(X\_train, y\_train)

    # check top performing n\_neighbors value

    print("\nBest Parameter: ")

    print(knn\_gscv.best\_params\_)

    # check the mean score for the top performing value of n\_neighbors

    print("\nBest Score: ")

    print(knn\_gscv.best\_score\_)

    print("\n#1\_2)Hyper Parameter tuning by RandomizedSearch")

    # create a new KNN model

    knn3 = KNeighborsClassifier()

    # create a dictionary of all values we want to test for n\_neighbors

    param\_grid = {'n\_neighbors': np.arange(1,25)}

    # use GridSearch to test all values for n\_neighbors

    knn\_gscv = RandomizedSearchCV(knn3, param\_grid, cv=5)

    # fit model to data

    knn\_gscv.fit(X\_train, y\_train)

    # check top performing n\_neighbors value

    print("\nBest Parameter: ")

    print(knn\_gscv.best\_params\_)

    # check the mean score for the top performing value of n\_neighbors

    print("\nBest Score: ")

    print(knn\_gscv.best\_score\_)

# enumerate splits (for k=5)

for train, test in kfold.split(others):

    X\_train = others[train]

    y\_train = quality[train]

    X\_test = others[test]

    y\_test = quality[test]

    # build a KNN model, create KNN classifier (k=5)

    knn\_cv = KNeighborsClassifier(n\_neighbors=5)

    # check accuracy of our model on the test data

    cv\_score = cross\_val\_score(knn\_cv, X\_test, y\_test, cv=5)

    print("#2) Evaluate the K-NN(k=5) model: ", cv\_scores)

    print("\n#2\_1)Hyper Parameter tuning by GridSearch")

    # create a new KNN model

    knn2 = KNeighborsClassifier()

    # create a dictionary of all values we want to test for n\_neighbors

    param\_grid = {'n\_neighbors': np.arange(1,25)}

    # use GridSearch to test all values for n\_neighbors

    knn\_gscv = GridSearchCV(knn2, param\_grid, cv=5)

    # fit model to data

    knn\_gscv.fit(X\_train, y\_train)

    # check top performing n\_neighbors value

    print("\nBest Parameter: ")

    print(knn\_gscv.best\_params\_)

    # check the mean score for the top performing value of n\_neighbors

    print("\nBest Score: ")

    print(knn\_gscv.best\_score\_)

    print("\n#2\_2)Hyper Parameter tuning by RandomizedSearch")

    # create a new KNN model

    knn3 = KNeighborsClassifier()

    # create a dictionary of all values we want to test for n\_neighbors

    param\_grid = {'n\_neighbors': np.arange(1,25)}

    # use GridSearch to test all values for n\_neighbors

    knn\_gscv = RandomizedSearchCV(knn3, param\_grid, cv=5)

    # fit model to data

    knn\_gscv.fit(X\_train, y\_train)

    # check top performing n\_neighbors value

    print("\nBest Parameter: ")

    print(knn\_gscv.best\_params\_)

    # check the mean score for the top performing value of n\_neighbors

    print("\nBest Score: ")

    print(knn\_gscv.best\_score\_)

- output -

텍스트이(가) 표시된 사진

자동 생성된 설명

(Time takes too long, so I’ll show partition of all.)