

## -ML 監督式學習

(Machine Learning, Supervised Learning, KNN)

Knn 模型設定以 k 值為 6，將模型 fit 到(X, y)

```
script.py
1 # Import KNeighborsClassifier
2 from sklearn.neighbors import KNeighborsClassifier
3
4 # Create arrays for the features and the target variable
5 y = churn_df["churn"].values
6 X = churn_df[["account_length", "customer_service_calls"]].values
7
8 # Create a KNN classifier with 6 neighbors
9 knn = KNeighborsClassifier(n_neighbors = 6)
10
11 # Fit the classifier to the data
12 knn.fit(X, y)
```

Run Code Submit Answer

```
IPython Shell Slides Notes
# Print the predictions for X_new
print("Predictions: {}".format(y_pred))

# Import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier

# Create arrays for the features and the target variable
y = churn_df["churn"].values
X = churn_df[["account_length", "customer_service_calls"]].values

# Create a KNN classifier with 6 neighbors
knn = KNeighborsClassifier(n_neighbors = 6)

# Fit the classifier to the data
knn.fit(X, y)

KNeighborsClassifier(n_neighbors=6)

In [1]:
```

帶入 y\_pred 得到預測結果  
0=未流失客戶，1=流失客戶

```
script.py
1 # Predict the labels for the X_new
2 y_pred = knn.predict(X_new)
3
4 # Print the predictions for X_new
5 print("Predictions: {}".format(y_pred))
```

Run Code Submit Answer

```
IPython Shell Slides Notes
# Create arrays for the features and the target variable
y = churn_df["churn"].values
X = churn_df[["account_length", "customer_service_calls"]].values

# Create a KNN classifier with 6 neighbors
knn = KNeighborsClassifier(n_neighbors = 6)

# Fit the classifier to the data
knn.fit(X, y)

# Predict the labels for the X_new
y_pred = knn.predict(X_new)

# Print the predictions for X_new
print("Predictions: {}".format(y_pred))

Predictions: [0 1 0]

In [1]:
```

## Model Measuring 模型準確度測試

以 knn 模型測試準確度

將 test 跟 train 從 dataset 中取出 20%，種子設定 42

Stratify 穩定 test 跟 train 的比例

```
script.py
1 # Import the module
2 from sklearn.model_selection import train_test_split
3
4 X = churn_df.drop("churn", axis=1).values
5 y = churn_df["churn"].values
6
7 # Split into training and test sets
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
9 knn = KNeighborsClassifier(n_neighbors=5)
10
11 # Fit the classifier to the training data
12 knn.fit(X_train, y_train)
13
14 # Print the accuracy
15 print(knn.score(X_test, y_test))
```

Python Shell Slides Notes

```
# Import the module
from sklearn.model_selection import train_test_split

X = churn_df.drop("churn", axis=1).values
y = churn_df["churn"].values

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the classifier to the training data
knn.fit(X_train, y_train)

# Print the accuracy
print(knn.score(X_test, y_test))
```

0.874629685157422

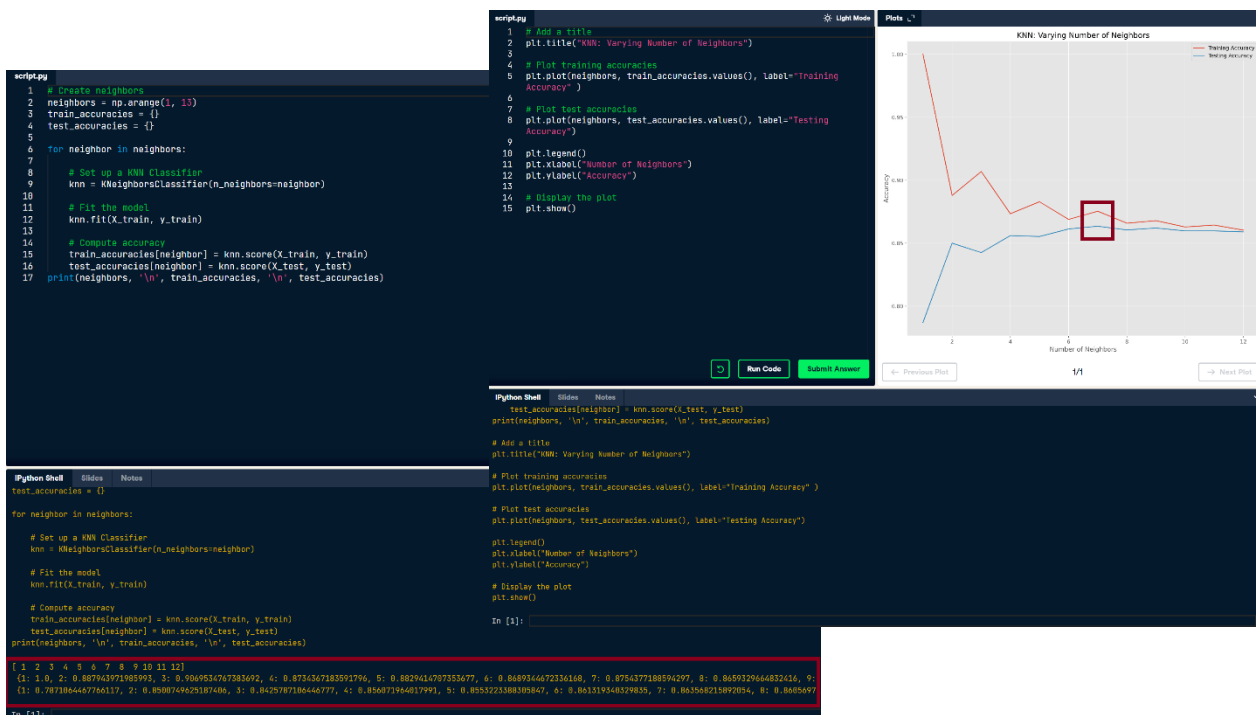
0.874 以 20% dataset 來說準確率並不高

或是用 Model Complexity 和 knn 模型訓練(左圖)

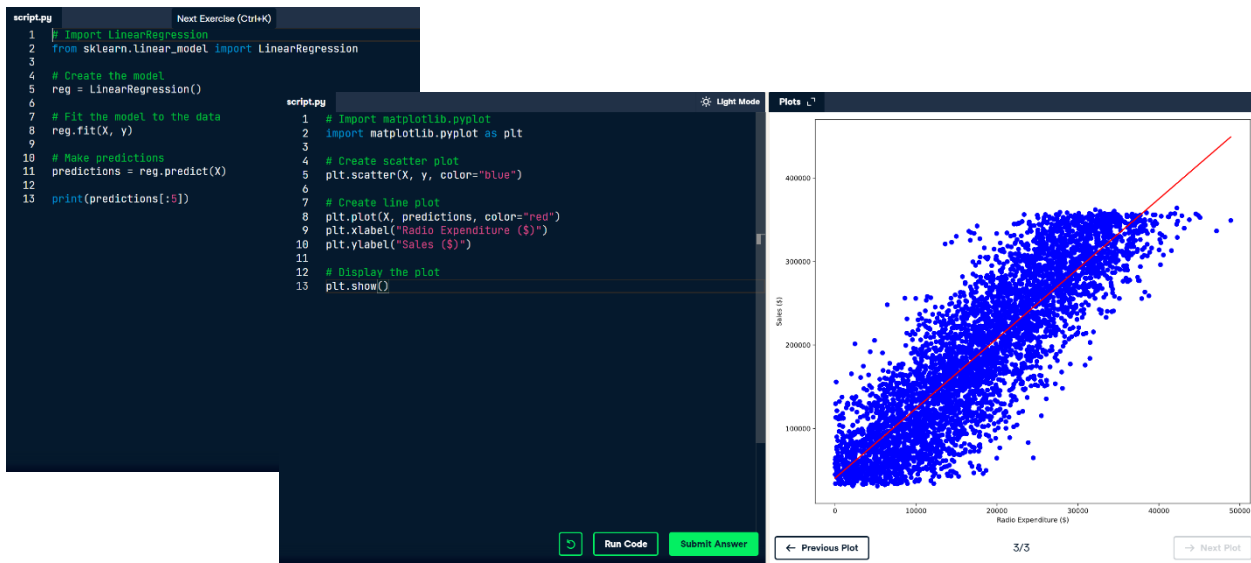
以 for loop 得出 X, y 準確率

而(右圖)視覺化後可以發現，K 值越大，過度擬合；K 值越小，則未擬合

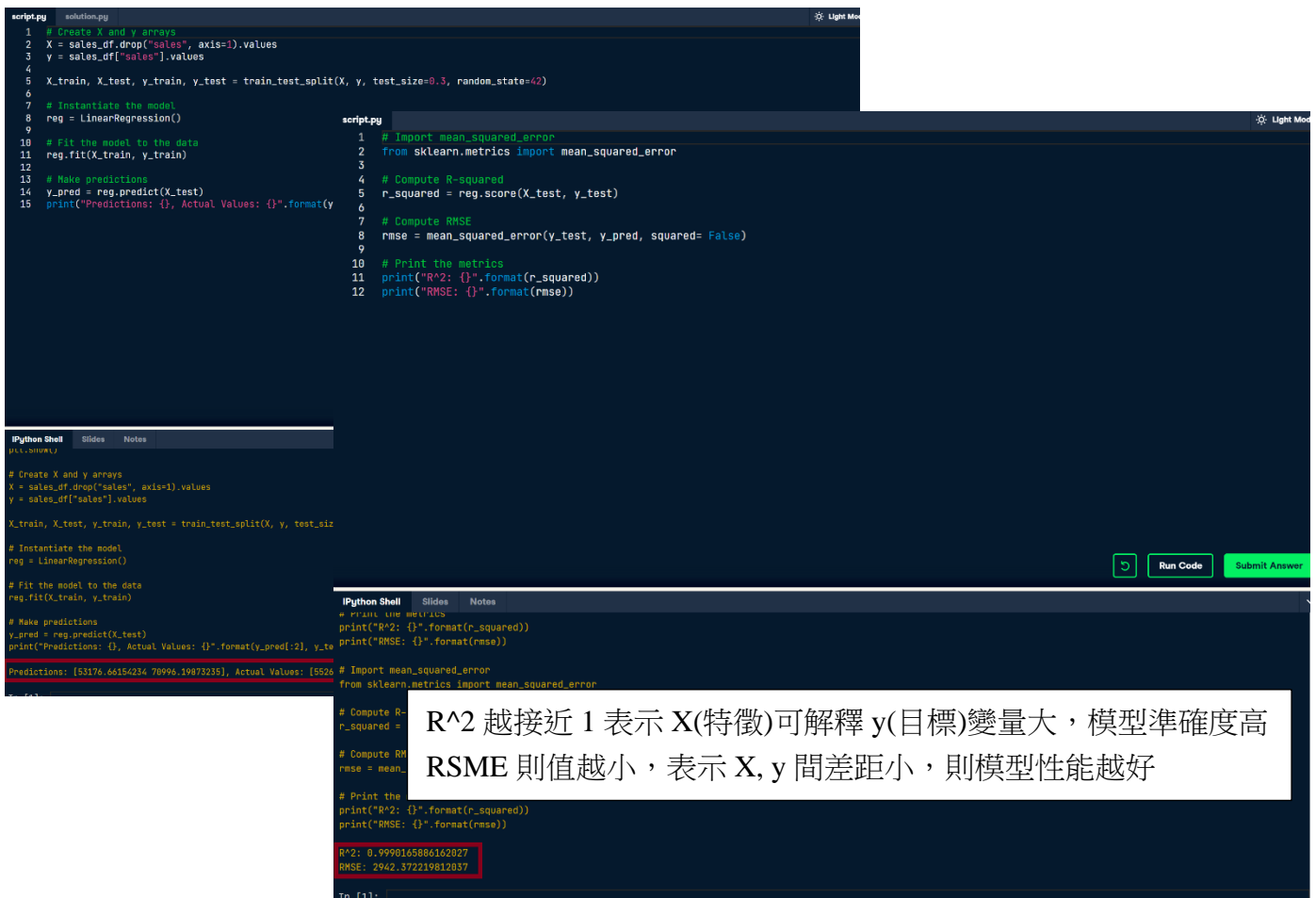
右圖中紅框即為最佳擬合點



將 X, y fit 到 LinearRegression 作為預測模型  
再用 plt.plot 視覺化觀察線性(預測)與散點圖(原值)差異



利用 .drop 分出特徵(X)及目標組(y)做預測  
再計算 R<sup>2</sup> 跟 RMSE 測試模型準確度



R<sup>2</sup> 越接近 1 表示 X(特徵)可解釋 y(目標)變量大，模型準確度高  
RMSE 則值越小，表示 X, y 間差距小，則模型性能越好

利用 KFold, cross\_val\_score 做交叉驗證 (cross validation)

```
script.py
1 # Import the necessary modules
2 from sklearn.model_selection import KFold, cross_val_score
3
4 # Create a KFold object
5 kf = KFold(n_splits=6, shuffle=True, random_state=5)
6
7 reg = LinearRegression()
8
9 # Compute 6-fold cross-validation scores
10 cv_scores = cross_val_score(reg, X, y, cv=kf)
11
12 # Print scores
13 print(cv_scores)
```

Ctrl+Shift+Enter

Run Code Submit Answer

```
IPython Shell Slides Notes
File <stdin>, line 13, in <module>
  print(scores)
NameError: name 'scores' is not defined

# Import the necessary modules
from sklearn.model_selection import KFold, cross_val_score

# Create a KFold object
kf = KFold(n_splits=6, shuffle=True, random_state=5)

reg = LinearRegression()

# Compute 6-fold cross-validation scores
cv_scores = cross_val_score(reg, X, y, cv=kf)

# Print scores
print(cv_scores)

[0.74451678 0.77241887 0.76842114 0.7418486 0.75178822 0.74486484]
```

## Regularized Regression 正規化模型

(Ridge, Lasso)

Ridge:

當  $\alpha$  值越大，模型過度擬合；反之，則未擬合 ( $\alpha$  類似 K 值)

```
script.py
1 # Import Ridge
2 from sklearn.linear_model import Ridge
3 alphas = [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
4 ridge_scores = []
5 for alpha in alphas:
6
7     # Create a Ridge regression model
8     ridge = Ridge(alpha=alpha)
9
10    # Fit the data
11    ridge.fit(X_train, y_train)
12
13    # Obtain R-squared
14    score = ridge.score(X_test, y_test)
15    ridge_scores.append(score)
16
17    print(ridge_scores)

Run Code Submit Answer

Python Shell Slides Notes
# Import Ridge
from sklearn.linear_model import Ridge
alphas = [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
ridge_scores = []
for alpha in alphas:

    # Create a Ridge regression model
    ridge = Ridge(alpha=alpha)

    # Fit the data
    ridge.fit(X_train, y_train)

    # Obtain R-squared
    score = ridge.score(X_test, y_test)
    ridge_scores.append(score)

print(ridge_scores)

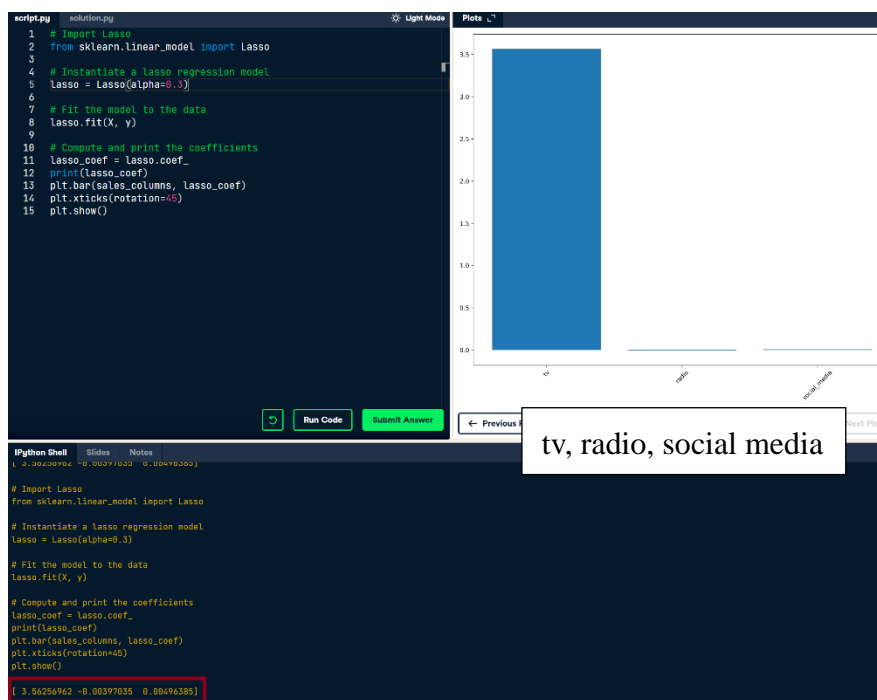
[0.9998152104759369, 0.9998152104759373, 0.9998152104759419, 0.9998152104759871, 0.9998152104764387, 0.9998152104809561]
```

Lasso:

Lasso 傾向把不重要的特徵係數縮小至 0，可用來驗證特徵 X 的重要性

圖中，電視的係數最高，電台與社群媒體則近於 0

得出電視為主要商品銷售平台



## ROC 模型可視化

可看出該模型的表現比原本隨機觀測的結果好很多  
(ROC 曲線在虛線上方)



# GridSearch & RandomizedSearch 網格搜尋&隨機搜尋

## 利用 GridSearch 調整超參數

```
script.py solution.py
1 # Import GridSearchCV
2 from sklearn.model_selection import GridSearchCV
3
4 # Set up the parameter grid
5 param_grid = {'alpha': np.linspace(0.00001, 1, 20)}
6
7 # Instantiate Lasso_cv
8 lasso_cv = GridSearchCV(lasso, param_grid, cv=kf)
9
10 # Fit to the training data
11 lasso_cv.fit(X_train, y_train)
12 print("Tuned lasso parameters: {}".format(lasso_cv.best_params_))
13 print("Tuned lasso score: {}".format(lasso_cv.best_score_))

Python Shell Slides Notes
Tuned lasso parameters: {'alpha': 1e-05}
Tuned lasso score: 0.53978807238121977

# Import GridSearchCV
from sklearn.model_selection import GridSearchCV

# Set up the parameter grid
param_grid = {'alpha': np.linspace(0.00001, 1, 20)}

# Instantiate Lasso_cv
lasso_cv = GridSearchCV(lasso, param_grid, cv=kf)

# Fit to the training data
lasso_cv.fit(X_train, y_train)
print("Tuned lasso parameters: {}".format(lasso_cv.best_params_))
print("Tuned lasso score: {}".format(lasso_cv.best_score_))

Tuned lasso parameters: {'alpha': 1e-05}
Tuned lasso score: 0.53978807238121977
```

如果資料量龐大，用 RandomizeSearch 調整超參數  
縮短學習時間

```
script.py solution.py
1 # Create the parameter space
2 params = {'penalty': ['l1', 'l2'],
3           'tol': np.linspace(0.0001, 1.0, 50),
4           'C': np.linspace(0.1, 1.0, 50),
5           'class_weight': ['balanced', {0:0.8, 1:0.2}]}
6
7 # Instantiate the RandomizedSearchCV object
8 logreg_cv = RandomizedSearchCV(logreg, params, cv=kf)
9
10 # Fit the data to the model
11 logreg_cv.fit(X_train, y_train)
12
13 # Print the tuned parameters and score
14 print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
15 print("Tuned Logistic Regression Best Accuracy Score: {}".format(logreg_cv.best_score_))

Python Shell Slides Notes
print("Tuned lasso score: {}".format(lasso_cv.best_score_))

# Create the parameter space
params = {'penalty': ['l1', 'l2'],
          'tol': np.linspace(0.0001, 1.0, 50),
          'C': np.linspace(0.1, 1.0, 50),
          'class_weight': ['balanced', {0:0.8, 1:0.2}]}

# Instantiate the RandomizedSearchCV object
logreg_cv = RandomizedSearchCV(logreg, params, cv=kf)

# Fit the data to the model
logreg_cv.fit(X_train, y_train)

# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Tuned Logistic Regression Best Accuracy Score: {}".format(logreg_cv.best_score_))

Tuned Logistic Regression Parameters: {'tol': 0.14294280714280712, 'penalty': 'l2', 'class_weight': 'balanced', 'C': 0.6326530612244898}
Tuned Logistic Regression Best Accuracy Score: 0.7460802633613221
```

## Dummy 轉換

Dummy 將資料轉換為 0 跟 1 (原為音樂種類)，以利於模型應用  
算出 RMSE & STD 評估模型

```
script.py
1 # Create X and y
2 X = music_dummies.drop('popularity', axis=1).values
3 y = music_dummies['popularity'].values
4
5 # Instantiate a ridge model
6 ridge = Ridge(alpha=0.2)
7
8 # Perform cross-validation
9 scores = cross_val_score(ridge, X, y, cv=kf, scoring="neg_mean_squared_error")
10
11 # Calculate RMSE
12 rmse = np.sqrt(-scores)
13 print("Average RMSE: {}".format(np.mean(rmse)))
14 print("Standard Deviation of the target array: {}".format(np.std(y)))
```

Ctrl+Shift+Enter

Run Code Submit Answer

```
IPython Shell Slides Notes
Average RMSE: 8.236853840202299
Standard Deviation of the target array: 14.02156909907019

# Create X and y
X = music_dummies.drop('popularity', axis=1).values
y = music_dummies['popularity'].values

# Instantiate a ridge model
ridge = Ridge(alpha=0.2)

# Perform cross-validation
scores = cross_val_score(ridge, X, y, cv=kf, scoring="neg_mean_squared_error")

# Calculate RMSE
rmse = np.sqrt(-scores)
print("Average RMSE: {}".format(np.mean(rmse)))
print("Standard Deviation of the target array: {}".format(np.std(y)))

Average RMSE: 8.236853840202299
Standard Deviation of the target array: 14.02156909907019
```



# Pipeline 管道模型

先設置 steps ( knn & imputer)

```
script.py solution.py
1 # import modules
2 from sklearn.impute import SimpleImputer
3 from sklearn.pipeline import Pipeline
4
5 # instantiate an imputer
6 imputer = SimpleImputer()
7
8 # instantiate a knn model
9 knn = KNeighborsClassifier(n_neighbors=3)
10
11 # Build steps for the pipeline
12 steps = [("imputer", imputer),
13         ("knn", knn)]
14
15 # Make predictions on the test set
16 y_pred = pipeline.predict(X_test)
17
18 # Print the confusion matrix
19 print(confusion_matrix(y_test, y_pred))
20
21 # Import modules
22 from sklearn.impute import SimpleImputer
23 from sklearn.pipeline import Pipeline
24
25 # Instantiate an imputer
26 imputer = SimpleImputer()
27
28 # Instantiate a knn model
29 knn = KNeighborsClassifier(n_neighbors=3)
30
31 # Build steps for the pipeline
32 steps = [("imputer", imputer),
33         ("knn", knn)]
```

steps 放到 Pipeline 裡

Confusion Matrix 自動處理(即 Pipeline)後，得出結果

```
script.py solution.py
1 steps = [("imputer", imp_mean),
2         ("knn", knn)]
3
4 # Create the pipeline
5 pipeline = Pipeline(steps)
6
7 # Fit the pipeline to the training data
8 pipeline.fit(X_train, y_train)
9
10 # Make predictions on the test set
11 y_pred = pipeline.predict(X_test)
12
13 # Print the confusion matrix
14 print(confusion_matrix(y_test, y_pred))
15
16 # Import modules
17 from sklearn.impute import SimpleImputer
18 from sklearn.pipeline import Pipeline
19
20 # Instantiate an imputer
21 imputer = SimpleImputer()
22
23 # Instantiate a knn model
24 knn = KNeighborsClassifier(n_neighbors=3)
25
26 # Build steps for the pipeline
27 steps = [("imputer", imputer),
28         ("knn", knn)]
29
30 # Create the pipeline
31 pipeline = Pipeline(steps)
32
33 # Fit the pipeline to the training data
34 pipeline.fit(X_train, y_train)
35
36 # Make predictions on the test set
37 y_pred = pipeline.predict(X_test)
38
39 # Print the confusion matrix
40 print(confusion_matrix(y_test, y_pred))
41
42 [[79  9]
43  [ 4 82]]
```

## Scale 模型縮放

在模型縮放前，score 為 0.619

```
script.py  solution.py
1 # Import StandardScaler
2 from sklearn.preprocessing import StandardScaler
3
4 # Create pipeline steps
5 steps = [StandardScaler(),
6          Lasso(alpha=0.5)]
7
8 # Instantiate the pipeline
9 pipeline = Pipeline(steps)
10 pipeline.fit(X_train, y_train)
11
12 # Calculate and print R-squared
13 print(pipeline.score(X_test, y_test))
```

Run all/selected code

Run Code Submit Answer

```
iPython Shell Slides Notes
y_pred = pipeline.predict(X_test)

# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))

# Import StandardScaler
from sklearn.preprocessing import StandardScaler

# Create pipeline steps
steps = [StandardScaler(),
        Lasso(alpha=0.5)]

# Instantiate the pipeline
pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)

# Calculate and print R-squared
print(pipeline.score(X_test, y_test))

0.6193523316282489
```

縮放後為 0.8425，準確率上升

```
script.py  solution.py
1 # Build the steps
2 steps = [StandardScaler(),
3          LogisticRegression()]
4 pipeline = Pipeline(steps)
5
6 # Create the parameter space
7 parameters = {"logreg__C": np.linspace(0.001, 1.0, 20)}
8 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
9                                                    random_state=21)
10
11 # Instantiate the grid search object
12 cv = GridSearchCV(pipeline, param_grid=parameters)
13
14 # Fit to the training data
15 cv.fit(X_train, y_train)
16 print(cv.best_score_, "\n", cv.best_params_)
```

Run Code Submit Answer

```
iPython Shell Slides Notes
# Build the steps
steps = [StandardScaler(),
        LogisticRegression()]
pipeline = Pipeline(steps)

# Create the parameter space
parameters = {"logreg__C": np.linspace(0.001, 1.0, 20)}
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=21)

# Instantiate the grid search object
cv = GridSearchCV(pipeline, param_grid=parameters)

# Fit to the training data
cv.fit(X_train, y_train)
print(cv.best_score_, "\n", cv.best_params_)

0.8425
{"logreg__C": 0.1061578947368421}
```

## Pipeline & Scale 最終組合

```
script.py solution.py
1 # Create steps
2 steps = [("imp_mean", SimpleImputer()),
3          ("scaler", StandardScaler()),
4          ("logreg", LogisticRegression())]
5
6 # Set up pipeline
7 pipeline = Pipeline(steps)
8 params = {"logreg__solver": ["newton-cg", "saga", "lbfgs"],
9          "logreg__C": np.linspace(0.001, 1.0, 10)}
10
11 # Create the GridSearchCV object
12 tuning = GridSearchCV(pipeline, param_grid=params)
13 tuning.fit(X_train, y_train)
14 y_pred = tuning.predict(X_test)
15
16 # Compute and print performance
17 print("Tuned Logistic Regression Parameters: {}, Accuracy: {}".format(tuning.best_params_, tuning.score(X_test, y_test)))
```

Run Code Submit Answer

```
IPython Shell Slides Notes
# Create steps
steps = [("imp_mean", SimpleImputer()),
          ("scaler", StandardScaler()),
          ("logreg", LogisticRegression())]

# Set up pipeline
pipeline = Pipeline(steps)
params = {"logreg__solver": ["newton-cg", "saga", "lbfgs"],
          "logreg__C": np.linspace(0.001, 1.0, 10)}

# Create the GridSearchCV object
tuning = GridSearchCV(pipeline, param_grid=params)
tuning.fit(X_train, y_train)
y_pred = tuning.predict(X_test)

# Compute and print performance
print("Tuned Logistic Regression Parameters: {}, Accuracy: {}".format(tuning.best_params_, tuning.score(X_test, y_test)))

Tuned Logistic Regression Parameters: {'logreg__C': 0.112, 'logreg__solver': 'newton-cg'}, Accuracy: 0.82
```