06_regression_cross_validation

May 7, 2025

[ ]:
```
import os
user = os.getenv('USER')
os.chdir(f'/scratch/cd82/{user}/notebooks')
```

0.1 Linear Regression - Cross Validation

Cross validation is a method for testing a model using all of the data available. It cycles through selection of a training and test data set, producing test metrics for each model. The test metrics are then used to confirm (or otherwise) that the samples are valid and the quality of the model.

There are many different ways to split the data and Scikit-Learn has a broad range of methods. These include: - KFold - GroupKFold - ShuffleSplit - StratifiedKFold - StratifiedGroupKFold - GroupShuffleSplit - StratifiedShuffleSplit - TimeSeriesSplit

An example of how kfold cross validation (with k = 5) can split a data set is shown here:

ref: sklearn.model_selection.cross_validate.html

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.metrics import mean_squared_error, r2_score
```

[2]:
```
# Generate sample data
N = 30  # the number of samples to be created

# The seed for the random number generator.
seed_seq = np.random.SeedSequence(42)
# Create a random number generator instance
rng = np.random.default_rng(seed_seq)

rndg = rng.normal(loc=0.0, scale=1, size=N)
rndg = rndg.reshape((N, 1))
print('rndg shape: ', rndg.shape)

# Create X data
start_x = 2.0
```

```
range_x = 2.0

X = np.linspace(start_x, start_x+range_x, num=N )
X = X.reshape((N, 1))
print('X shape: ', X.shape)

pc_rand = 0.75  # +- how much randomness to be added to y data

# Create y data
offset_y = 6.0
slope_y = 4.0

add_offset = (start_x * slope_y) + offset_y
# add_offset =  offset_y
y = (( add_offset )+ slope_y * (X - start_x)) + (pc_rand * rndg)
print('y shape: ',y.shape)
```

```
rndg shape:  (30, 1)
X shape:  (30, 1)
y shape:  (30, 1)
```

Choices for test metrics The created sub-models produced by cross validation can be scored using different metrics. Choices are: ||| |———————————-|———————————| |R^2 | 'r2' | |Mean Absolute Error (MAE) | 'neg_mean_absolute_error' | |Mean Squared Error (MSE) | 'neg_mean_squared_error' | |Root Mean Squared Error (RMSE) | 'neg_root_mean_squared_error'| |Mean Absolute Percentage Error (MAPE)| 'neg_mean_absolute_percentage_error'|

[3]:
```
# Train the model
model = LinearRegression()
scores = cross_val_score(model, X,y, cv=5, scoring='r2')

print("Cross-validation scores: ",scores)
# model.fit(X_train, y_train)
```

```
Cross-validation scores:  [-0.45584705  0.63990097 -0.4162912   0.37711775
0.84601569]
```

[4]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate

# The shuffle parameter is important if data has some internal structure
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# kf = KFold(n_splits=5, shuffle=False)
kf.get_n_splits(X)

print(kf)
```

```python
# This code prints what samples are in either the test or train split for each␣
 ↪fold
for i, (train_index, test_index) in enumerate(kf.split(X)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")

results = cross_validate(model, X, y, cv=kf,
                         scoring=('r2', 'neg_mean_squared_error'),
                         return_train_score=True)

print("Train r^2 scores:", results['train_r2'])
print("Test r^2 scores:", results['test_r2'])

print("Train mse scores:", results['train_neg_mean_squared_error'])
print("Test mse scores:", results['test_neg_mean_squared_error'])
```

```
KFold(n_splits=5, random_state=42, shuffle=True)
Fold 0:
  Train: index=[ 0  1  2  3  4  5  6  7 10 11 12 13 14 16 18 19 20 21 22 24 25
26 28 29]
  Test:  index=[ 8  9 15 17 23 27]
Fold 1:
  Train: index=[ 1  2  3  5  6  7  8  9 10 11 13 14 15 17 18 19 20 21 22 23 25
26 27 29]
  Test:  index=[ 0  4 12 16 24 28]
Fold 2:
  Train: index=[ 0  3  4  6  7  8  9 10 12 14 15 16 17 18 19 20 21 23 24 25 26
27 28 29]
  Test:  index=[ 1  2  5 11 13 22]
Fold 3:
  Train: index=[ 0  1  2  4  5  6  7  8  9 10 11 12 13 14 15 16 17 19 20 22 23
24 27 28]
  Test:  index=[ 3 18 21 25 26 29]
Fold 4:
  Train: index=[ 0  1  2  3  4  5  8  9 11 12 13 15 16 17 18 21 22 23 24 25 26
27 28 29]
  Test:  index=[ 6  7 10 14 19 20]
Train r^2 scores: [0.95345323 0.94888535 0.95827095 0.94587824 0.952061  ]
Test r^2 scores: [0.93087285 0.95660944 0.90839863 0.93497167 0.93763753]
Train mse scores: [-0.32831874 -0.29915417 -0.24800092 -0.32227466 -0.36335646]
Test mse scores: [-0.30696172 -0.39895631 -0.61469978 -0.31104276 -0.13715491]
```

Respect groupings in the dataset It is possible to respect groupings of samples in cross-validation by using the `GroupKFold` cross-validator

[5]:
```
from sklearn.model_selection import GroupKFold

# Set an array to specify what sample belongs to what group
groups = [i // 5 for i in range(N)]
print('Groups = ', groups)
group_kfold = GroupKFold(n_splits=4, shuffle=True, random_state=42)

# Perform cross-validation
results = cross_validate(model, X, y, cv=group_kfold,
                         groups=groups,
                         scoring=('r2', 'neg_mean_squared_error'),
                         return_train_score=True)

for i, (train_index, test_index) in enumerate(group_kfold.split(X,␣
 ↪groups=groups)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")

print("Train scores:", results['train_r2'])
print("Test scores:", results['test_r2'])
```

```
Groups =  [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4,
4, 4, 5, 5, 5, 5, 5]
Fold 0:
  Train: index=[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29]
  Test:  index=[0 1 2 3 4 5 6 7 8 9]
Fold 1:
  Train: index=[ 0  1  2  3  4  5  6  7  8  9 15 16 17 18 19 20 21 22 23 24]
  Test:  index=[10 11 12 13 14 25 26 27 28 29]
Fold 2:
  Train: index=[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 25
26 27 28
 29]
  Test:  index=[20 21 22 23 24]
Fold 3:
  Train: index=[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 20 21 22 23 24 25
26 27 28
 29]
  Test:  index=[15 16 17 18 19]
Train scores: [0.91535751 0.9367356  0.95216088 0.9580737 ]
Test scores: [0.11072382 0.92326017 0.28835825 0.50758492]
```

Detect outliers in the data

[6]:
```
from sklearn.model_selection import cross_val_predict
from scipy.stats import zscore
```

```python
# Create an outlier in the data
y[17] = y[16] + 3.0

# kf = KFold(n_splits=5, shuffle=False, random_state=None )
kf = KFold(n_splits=5, shuffle=True, random_state=42)
y_pred = cross_val_predict(model, X, y, cv=kf)

for i, (train_index, test_index) in enumerate(kf.split(X)):
    print(f"Fold {i}:")
    print(f"  Train: index={train_index}")
    print(f"  Test:  index={test_index}")

# print(predictions)
# Calculate residuals
residuals = y - y_pred

# Detect outliers using z-score (standardised by mean and sd)
z_scores = zscore(residuals)

# This test finds data points that are more than 3 standard
# deviations from the mean indicating they are probable outliers
outliers = np.where(np.abs(z_scores) > 3)[0]

print("Outliers detected at indices:", outliers)
```

```
Fold 0:
  Train: index=[ 0  1  2  3  4  5  6  7 10 11 12 13 14 16 18 19 20 21 22 24 25
26 28 29]
  Test:  index=[ 8  9 15 17 23 27]
Fold 1:
  Train: index=[ 1  2  3  5  6  7  8  9 10 11 13 14 15 17 18 19 20 21 22 23 25
26 27 29]
  Test:  index=[ 0  4 12 16 24 28]
Fold 2:
  Train: index=[ 0  3  4  6  7  8  9 10 12 14 15 16 17 18 19 20 21 23 24 25 26
27 28 29]
  Test:  index=[ 1  2  5 11 13 22]
Fold 3:
  Train: index=[ 0  1  2  4  5  6  7  8  9 10 11 12 13 14 15 16 17 19 20 22 23
24 27 28]
  Test:  index=[ 3 18 21 25 26 29]
Fold 4:
  Train: index=[ 0  1  2  3  4  5  8  9 11 12 13 15 16 17 18 21 22 23 24 25 26
27 28 29]
  Test:  index=[ 6  7 10 14 19 20]
Outliers detected at indices: [17]
```

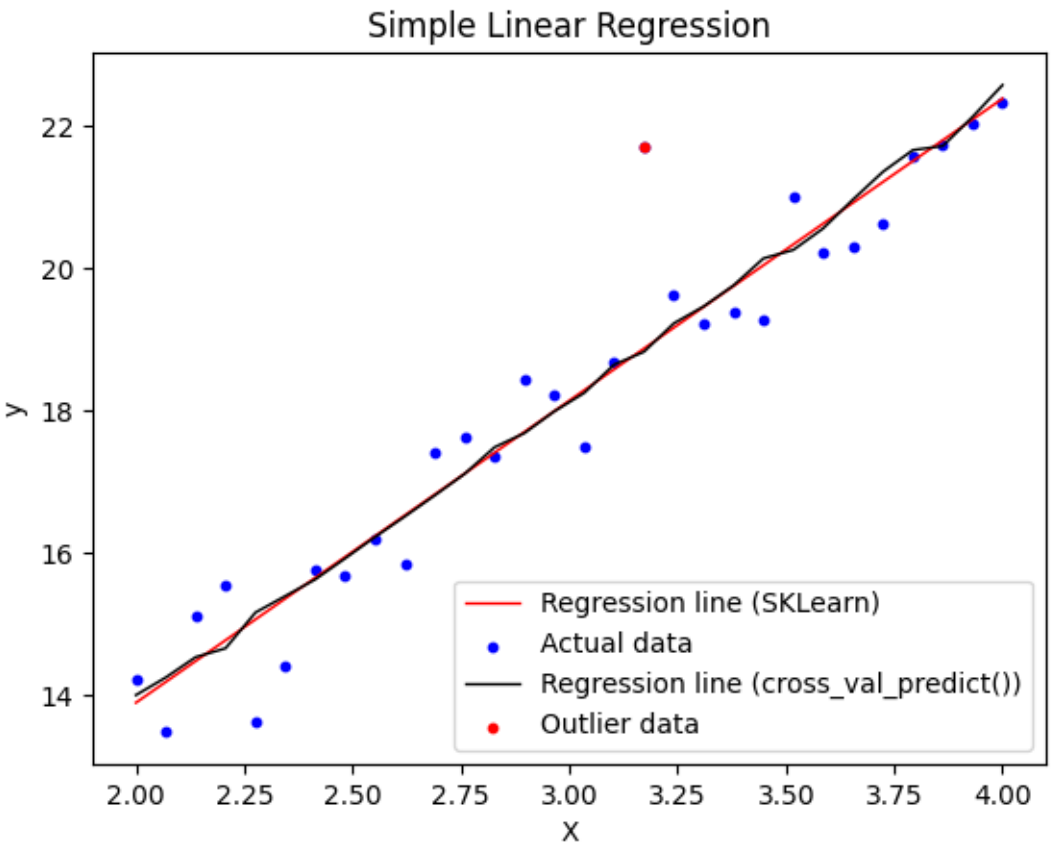
[7]:
```
# Evaluate the model
mse = mean_squared_error(y, y_pred)
r2 = r2_score(y, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R² Score: {r2}")
```

```
Mean Squared Error: 0.6119371881541689
R² Score: 0.9123238601264881
```

[8]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
# Train the model
model_skl = LinearRegression()
model_skl.fit(X, y)
```

[8]:
```
LinearRegression()
```

[9]:
```
y_pred_skl = model_skl.predict(X)


plt.plot(X, y_pred_skl, color='red',
         linewidth=1, label='Regression line (SKLearn)')

# Plot the results
plt.scatter(X, y, s=10, color='blue', label='Actual data')

plt.plot(X, y_pred, color='black',
         linewidth=1, label='Regression line (cross_val_predict())')

plt.scatter(X[outliers], y[outliers], s=10, color='red', label='Outlier data')

plt.xlabel('X')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```


Simple Linear Regression
22
20
18
16
14
y
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00
X
Regression line (SKLearn)
Actual data
Regression line (cross_val_predict())
Outlier data

[ ]: