00-Cross-Validation

February 13, 2023

---

1 Introduction to Cross Validation

In this lecture series we will do a much deeper dive into various methods of cross-validation. As well as a discussion on the general philosphy behind cross validation. A nice official documentation guide can be found here: https://scikit-learn.org/stable/modules/cross_validation.html

1.1 Imports

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

1.2 Data Example

[2]:
```
df = pd.read_csv("../DATA/Advertising.csv")
```

[3]:
```
df.head()
```

[3]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

1.3 —-

1.4 Train | Test Split Procedure

0. Clean and adjust data as necessary for X and y
1. Split Data in Train/Test for both X and y
2. Fit/Train Scaler on Training X Data
3. Scale X Test Data
4. Create Model
5. Fit/Train Model on X Train Data
6. Evaluate Model on X Test Data (by creating predictions and comparing to Y_test)
7. Adjust Parameters as Necessary and repeat steps 5 and 6

[6]:
```
## CREATE X and y
X = df.drop('sales',axis=1)
y = df['sales']

# TRAIN TEST SPLIT
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=101)

# SCALE DATA as all the features may not have same unit
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Create Model

[7]:
```
from sklearn.linear_model import Ridge
```

[8]:
```
# Poor Alpha Choice on purpose!
model = Ridge(alpha=100)
```

[9]:
```
model.fit(X_train,y_train)
```

[9]:
```
Ridge(alpha=100)
```

[10]:
```
y_pred = model.predict(X_test)
```

Evaluation

[11]:
```
from sklearn.metrics import mean_squared_error
```

[12]:
```
np.mean(df['sales'],axis=0)
```

[12]:
```
14.0225
```

[13]:
```
mean_squared_error(y_test,y_pred)   #that's a lot error in the predicted values␣
↪on comparing with mean of sale
```

[13]:
```
7.34177578903413
```

Adjust Parameters and Re-evaluate

[14]:
```
model = Ridge(alpha=1)
```

[15]:
```
model.fit(X_train,y_train)
```

[15]:
```
Ridge(alpha=1)
```

[16]:
```
y_pred = model.predict(X_test)
```

Another Evaluation

[17]:
```
mean_squared_error(y_test,y_pred)
```

[17]:
```
2.3190215794287514
```

Much better! We could repeat this until satisfied with performance metrics. (We previously showed RidgeCV can do this for us, but the purpose of this lecture is to generalize the CV process for any model).

1.5 —-

1.6 Train | Validation | Test Split Procedure

This is often also called a "hold-out" set, since you should not adjust parameters based on the final test set, but instead use it *only* for reporting final expected performance.

0. Clean and adjust data as necessary for X and y
1. Split Data in Train/Validation/Test for both X and y
2. Fit/Train Scaler on Training X Data
3. Scale X Eval Data
4. Create Model
5. Fit/Train Model on X Train Data
6. Evaluate Model on X Evaluation Data (by creating predictions and comparing to Y_eval)
7. Adjust Parameters as Necessary and repeat steps 5 and 6
8. Get final metrics on Test set (not allowed to go back and adjust after this!)

[18]:
```
## CREATE X and y
X = df.drop('sales',axis=1)
y = df['sales']
```

[19]:
```
#########################################################################
#### SPLIT TWICE! Here we create TRAIN | VALIDATION | TEST  #########
```

```
##########################################################################
from sklearn.model_selection import train_test_split

# 70% of data is training data, set aside other 30%
X_train, X_OTHER, y_train, y_OTHER = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=101)

# Remaining 30% is split into evaluation and test sets
# Each is 15% of the original data size
X_eval, X_test, y_eval, y_test = train_test_split(X_OTHER, y_OTHER, test_size=0.
 ↪5, random_state=101)
```

[20]:
```
# SCALE DATA
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_eval = scaler.transform(X_eval)
X_test = scaler.transform(X_test)
```

Create Model

[21]:
```
from sklearn.linear_model import Ridge
```

[22]:
```
# Poor Alpha Choice on purpose!
model = Ridge(alpha=100)
```

[23]:
```
model.fit(X_train,y_train)
```

[23]:
```
Ridge(alpha=100)
```

[24]:
```
y_eval_pred = model.predict(X_eval)
```

Evaluation

[25]:
```
from sklearn.metrics import mean_squared_error
```

[26]:
```
mean_squared_error(y_eval,y_eval_pred)
```

[26]:
```
7.320101458823871
```

Adjust Parameters and Re-evaluate

[27]:
```
model = Ridge(alpha=1)
```

[28]:
```
model.fit(X_train,y_train)
```

[28]:
```
Ridge(alpha=1)
```

[29]:
```
y_eval_pred = model.predict(X_eval)
```

Another Evaluation

[30]:
```
mean_squared_error(y_eval,y_eval_pred)
```

[30]:
```
2.383783075056986
```

Final Evaluation (Can no longer edit parameters after this!)

[31]:
```
y_final_test_pred = model.predict(X_test)
```

[32]:
```
mean_squared_error(y_test,y_final_test_pred)
```

[32]:
```
2.254260083800518
```

1.7 —-

Cross Validation with cross_val_score

---

[33]:
```
## CREATE X and y
X = df.drop('sales',axis=1)
y = df['sales']

# TRAIN TEST SPLIT
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=101)

# SCALE DATA
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[50]:
```
model = Ridge(alpha=100)
```

[51]:
```
from sklearn.model_selection import cross_val_score
```

[52]:
```
    # SCORING OPTIONS:
    # https://scikit-learn.org/stable/modules/model_evaluation.html

scores = cross_val_score(model,X_train,y_train,
                         scoring='neg_mean_squared_error',cv=5)
```

[53]:
```
scores   # higher is better for negative mean squared error
         # -4 is better than -11
```

[53]:
```
array([ -9.32552967,  -4.9449624 , -11.39665242,  -7.0242106 ,
        -8.38562723])
```

[41]:
```
# Average of the MSE scores (we set back to positive)
abs(scores.mean())
```

[41]:
```
8.215396464543609
```

Adjust model based on metrics

[54]:
```
model = Ridge(alpha=1)
```

[55]:
```
# SCORING OPTIONS:
# https://scikit-learn.org/stable/modules/model_evaluation.html
scores = cross_val_score(model,X_train,y_train,
                         scoring='neg_mean_squared_error',cv=5)
```

[56]:
```
# Average of the MSE scores (we set back to positive)
abs(scores.mean())      # much better result here as compared to last one when␣
 ↪alpha was 100
```

[56]:
```
3.344839296530696
```

Final Evaluation (Can no longer edit parameters after this!)

[57]:
```
# Need to fit the model first!
model.fit(X_train,y_train)
```

[57]:
```
Ridge(alpha=1)
```

[58]:
```
y_final_test_pred = model.predict(X_test)
```

[59]:
```
mean_squared_error(y_test,y_final_test_pred)
```

[59]:
```
2.3190215794287514
```

1.8 —-

---

2 Cross Validation with cross_validate

The cross_validate function differs from cross_val_score in two ways:

It allows specifying multiple metrics for evaluation.

It returns a dict containing fit-times, score-times (and optionally training scores as well as fitted estimators) in addition to the test score.

For single metric evaluation, where the scoring parameter is a string, callable or None, the keys will be:

```
- ['test_score', 'fit_time', 'score_time']
```

And for multiple metric evaluation, the return value is a dict with the following keys:

```
['test_<scorer1_name>', 'test_<scorer2_name>', 'test_<scorer...>', 'fit_time', 'score_time']
```

return_train_score is set to False by default to save computation time. To evaluate the scores on the training set as well you need to be set to True.

[60]:
```
## CREATE X and y
X = df.drop('sales',axis=1)
y = df['sales']

# TRAIN TEST SPLIT
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=101)

# SCALE DATA
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

[61]:
```
model = Ridge(alpha=100)
```

[62]:
```
from sklearn.model_selection import cross_validate
```

[63]:
```
        # SCORING OPTIONS:
        # https://scikit-learn.org/stable/modules/model_evaluation.html

scores = cross_validate(model,X_train,y_train,
                        ␣
 ↪scoring=['neg_mean_absolute_error','neg_mean_squared_error','max_error'],cv=5)
```

[64]:
```
scores
```

[64]:
```
{'fit_time': array([0.00716829, 0.0030663 , 0.00267434, 0.00282359,
0.00250745]),
 'score_time': array([0.00344872, 0.01108909, 0.00368547, 0.0024929 ,
0.00268674]),
 'test_neg_mean_absolute_error': array([-2.31243044, -1.74653361, -2.56211701,
-2.01873159, -2.27951906]),
 'test_neg_mean_squared_error': array([ -9.32552967,  -4.9449624 , -11.39665242,
-7.0242106 ,
        -8.38562723]),
 'test_max_error': array([ -6.44988486,  -5.58926073, -10.33914027,
-6.61950405,
        -7.75578515])}
```

[65]:
```
pd.DataFrame(scores)
```

[65]:

	fit_time	score_time	test_neg_mean_absolute_error	test_neg_mean_squared_error	test_max_error
0	0.007168	0.003449	-2.312430	-9.325530	-6.449885
1	0.003066	0.011089	-1.746534	-4.944962	-5.589261
2	0.002674	0.003685	-2.562117	-11.396652	-10.339140
3	0.002824	0.002493	-2.018732	-7.024211	-6.619504
4	0.002507	0.002687	-2.279519	-8.385627	-7.755785

[66]:
```
pd.DataFrame(scores).mean()
```

[66]:
```
fit_time                      0.003648
score_time                    0.004681
test_neg_mean_absolute_error -2.183866
test_neg_mean_squared_error  -8.215396
test_max_error               -7.350715
dtype: float64
```

Adjust model based on metrics

[67]:
```
model = Ridge(alpha=1)
```

[68]:
```
# SCORING OPTIONS:
# https://scikit-learn.org/stable/modules/model_evaluation.html
scores = cross_validate(model,X_train,y_train,
                        ␣
 ↪scoring=['neg_mean_absolute_error','neg_mean_squared_error','max_error'],cv=5)
```

[69]:
```
pd.DataFrame(scores).mean()
```

[69]:
```
fit_time                      0.004222
score_time                    0.004109
test_neg_mean_absolute_error -1.319685
test_neg_mean_squared_error  -3.344839
test_max_error               -5.161145
dtype: float64
```

Final Evaluation (Can no longer edit parameters after this!)

[70]:
```
# Need to fit the model first!
model.fit(X_train,y_train)
```

[70]:
```
Ridge(alpha=1)
```

[71]:
```
y_final_test_pred = model.predict(X_test)
```

[72]:
```
mean_squared_error(y_test,y_final_test_pred)
```

[72]:
```
2.3190215794287514
```

2.1 —-