index

January 26, 2022

1 Introduction to Cross-Validation - Lab

1.1 Introduction

In this lab, you'll be able to practice your cross-validation skills!

1.2 Objectives

You will be able to:

- Perform cross validation on a model to determine optimal model performance
- Compare training and testing errors to determine if model is over or underfitting

1.3 Let's get started

We included the code to pre-process below.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

ames = pd.read_csv('ames.csv')

continuous = ['LotArea', '1stFlrSF', 'GrLivArea', 'SalePrice']
categoricals = ['BldgType', 'KitchenQual', 'SaleType', 'MSZoning',
                'Street', 'Neighborhood']

ames_cont = ames[continuous]

# log features
log_names = [f'{column}_log' for column in ames_cont.columns]

ames_log = np.log(ames_cont)
ames_log.columns = log_names

# normalize (subract mean and divide by std)

def normalize(feature):
    return (feature - feature.mean()) / feature.std()
```

```
ames_log_norm = ames_log.apply(normalize)

# one hot encode categoricals
ames_ohe = pd.get_dummies(ames[categoricals], prefix=categoricals,␣
 ↪drop_first=True)

preprocessed = pd.concat([ames_log_norm, ames_ohe], axis=1)

X = preprocessed.drop('SalePrice_log', axis=1)
y = preprocessed['SalePrice_log']
```

1.3.1 Train-test split

Perform a train-test split with a test set of 20%.

[2]:
```
# Import train_test_split from sklearn.model_selection
from sklearn.model_selection import train_test_split
```

[3]:
```
# Split the data into training and test sets (assign 20% to test set)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

[4]:
```
# A brief preview of train-test split
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
1168 292 1168 292
```

1.3.2 Fit the model

Fit a linear regression model and apply the model to make predictions on test set

[5]:
```
# Your code here
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)


y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

1.3.3 Residuals and MSE

Calculate the residuals and the mean squared error on the test set

[6]:
```
# Your code here
from sklearn.metrics import mean_squared_error

train_mse = mean_squared_error(y_train, y_train_pred)
```

```
test_mse = mean_squared_error(y_test, y_test_pred)

print('Train Mean Squarred Error:', train_mse)
print('Test Mean Squarred Error:', test_mse)
```

```
Train Mean Squarred Error: 0.15958730745079988
Test Mean Squarred Error: 0.1738025191908065
```

1.4 Cross-Validation: let's build it from scratch!

1.4.1 Create a cross-validation function

Write a function `kfolds()` that splits a dataset into k evenly sized pieces. If the full dataset is not divisible by k, make the first few folds one larger then later ones.

We want the folds to be a list of subsets of data!

[10]:
```
12 - 12%5
dk = 10/5
list(range(0,10, int(dk)))
```

[10]: `[0, 2, 4, 6, 8]`

[28]:
```
def kfolds(data, k):
    # Force data as pandas DataFrame
    data = pd.DataFrame(data)
    folds_dic = {}
    data_len = len(data)
    folds = []
    dk = data_len/k
    if data_len%k == 0:
        for i,item in enumerate(range(0,data_len, int(dk))):
            d = data.iloc[item:item+int(dk)]
            folds.append(d)
            folds_dic[i] = data.iloc[item:item+int(dk)]
    else:
        d_l = data_len - data_len%d
        dk = d_l/k
        for i, item in enumerate(range(0,dl, dk)):
            d = data.iloc[item:item+dk]
            folds.append(d)
            folds_dic[i] = data.iloc[item:item+dk]
            j = i
        folds.append(data.iloc[d_l:data_len])
#         folds_dic = pd.DataFrame(folds)
        folds_dic[j+1] = data.iloc[-j:]


    # add 1 to fold size to account for leftovers
```

```
    return folds_dic
```

1.4.2 Apply it to the Ames Housing data

[29]:
```
# Make sure to concatenate the data again
ames_data  = pd.concat([X.reset_index(drop=True), y], axis=1)
```

[32]:
```
# Apply kfolds() to ames_data with 5 folds
folds = kfolds(ames_data, 5)
```

1.4.3 Perform a linear regression for each fold and calculate the training and test error

Perform linear regression on each and calculate the training and test error:

[62]:
```
from sklearn.linear_model import LinearRegression

model_I = LinearRegression()



test_errs = []
train_errs = []
mean_both = []

k=5

# X = preprocessed.drop('SalePrice_log', axis=1)
# y = preprocessed['SalePrice_log']

for n in range(k):
    l = list(range(k))
    # Split in train and test for the fold
#     train = folds[n]
#     l.remove(n)
#     test_0 = folds[l[0]].append(folds[l[1]], ignore_index=True)
#     test_1 = test_0.append(folds[l[2]], ignore_index=True)
#     test = test_1.append(folds[l[3]], ignore_index=True)

    test = folds[n]
    l.remove(n)
    train_0 = folds[l[0]].append(folds[l[1]], ignore_index=True)
    train_1 = train_0.append(folds[l[2]], ignore_index=True)
    train = train_1.append(folds[l[3]], ignore_index=True)

#     test = folds[n:]
    model_I = LinearRegression()
    X_train = train.drop("SalePrice_log", axis = 1)
```

```
    y_train = train["SalePrice_log"]
    model_I.fit(X_train, y_train)
    y_train_pred = model_I.predict(X_train)

    X_test  = test.drop("SalePrice_log", axis = 1)
    y_test  = test["SalePrice_log"]
    y_test_pred = model_I.predict(X_test)




    # Evaluate Train and Test errors
    res_train = y_train - y_train_pred
    train_errs.append(np.mean(res_train**2))

    res_test = y_test - y_test_pred
    test_errs.append(np.mean(res_test**2))

    errs = (test_errs[n] + train_errs[n])

    mean_both.append(errs)

print(train_errs)
print(test_errs)
```

```
[0.1717050965146466, 0.15507935685930538, 0.15659946326223304,
0.16134557666308721, 0.1516504855313168]
[0.1243154614843743, 0.19350064631313132, 0.18910530431311173,
0.17079325250026917, 0.20742704588916958]
```

1.5 From GitHub

[50]:
```
def kfolds(data, k):
    # Force data as pandas DataFrame
    data = pd.DataFrame(data)
    num_observations = len(data)
    fold_size = num_observations//k
    leftovers = num_observations%k
    folds = []
    start_obs = 0
    for fold_n in range(1,k+1):
        if fold_n <= leftovers:
            #Fold Size will be 1 larger to account for leftovers
            fold =  data.iloc[start_obs : start_obs+fold_size+1]
            folds.append(fold)
            start_obs +=  fold_size + 1
        else:
```

```
            fold =  data.iloc[start_obs : start_obs+fold_size]
            folds.append(fold)
            start_obs +=  fold_size

    return folds
```

1.5.1 Apply it to the Ames Housing data

[51]:
```
ames_data = pd.concat([X.reset_index(drop=True), y], axis=1)
ames_folds = kfolds(ames_data, 5)
```

1.6 Perform a linear regression for each fold and calculate the training and test error

Perform linear regression on each and calculate the training and test error:

[53]:
```
test_errs = []
train_errs = []
k=5
linreg = LinearRegression()

for n in range(k):
    # Split in train and test for the fold
    train = pd.concat([fold for i, fold in enumerate(ames_folds) if i!=n])
    test = ames_folds[n]
    # Fit a linear regression model
    linreg.fit(X_train, y_train)
    #Evaluate Train and Test Errors
    y_hat_train = linreg.predict(X_train)
    y_hat_test = linreg.predict(X_test)
    train_residuals = y_hat_train - y_train
    test_residuals = y_hat_test - y_test
    train_errs.append(np.mean(train_residuals.astype(float)**2))
    test_errs.append(np.mean(test_residuals.astype(float)**2))
print(train_errs)
print(test_errs)
```

```
[0.16883836944959565, 0.16883836944959565, 0.16883836944959565,
0.16883836944959565, 0.16883836944959565]
[0.1902787622071821, 0.1902787622071821, 0.1902787622071821, 0.1902787622071821,
0.1902787622071821]
```

1.7 Cross-Validation using Scikit-Learn

This was a bit of work! Now, let's perform 5-fold cross-validation to get the mean squared error through scikit-learn. Let's have a look at the five individual MSEs and explain what's going on.

[47]:
```python
# Your code here
from sklearn.metrics import mean_squared_error, make_scorer
from sklearn.model_selection import cross_val_score

mse = make_scorer(mean_squared_error)

cv_5_results = cross_val_score(model, X, y, cv=5, scoring=mse)
```

[48]:
```python
cv_5_results
```

[48]:
```
array([0.12431546, 0.19350065, 0.1891053 , 0.17079325, 0.20742705])
```

[64]:
```python
print(np.round(test_errs,8))
```

```
[0.12431546 0.19350065 0.1891053  0.17079325 0.20742705]
```

[67]:
```python
 np.round(cv_5_results,8) - np.round(test_errs,8)
```

[67]:
```
array([0., 0., 0., 0., 0.])
```

Next, calculate the mean of the MSE over the 5 cross-validation and compare and contrast with the result from the train-test split case.

[68]:
```python
# Your code here
cv_5_results.mean()
```

[68]:
```
0.17702834210001123
```

1.8 Summary

Congratulations! You are now familiar with cross-validation and know how to use `cross_val_score()`. Remember that the results obtained from cross-validation are robust and always use it whenever possible!