learn-co-curriculum / dsc-cross-validation-lab Public

README.md

Introduction to Cross-Validation - Lab

Introduction

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

Objectives

You will be able to:

- Perform cross validation on a model
- Compare and contrast model validation strategies

Let's Get Started

We included the code to pre-process the Ames Housing dataset below. This is done for the sake of expediency, although it may result in data leakage and therefore overly optimistic model metrics.

```
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', 'Neighb

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']
```

Train-Test Split

Perform a train-test split with a test set of 20% and a random state of 4.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
```

Fit a Model

Fit a linear regression model on the training set

```
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

```
LinearRegression()
```

Calculate MSE

Calculate the mean squared error on the test set

```
from sklearn.metrics import mean_squared_error

y_hat_test = linreg.predict(X_test)
test_mse = mean_squared_error(y_test, y_hat_test)
test_mse
```

```
0.15233997210708167
```

Cross-Validation using Scikit-Learn

Now let's compare that single test MSE to a cross-validated test MSE.

```
from sklearn.model_selection import cross_val_score

cv_5_results = -cross_val_score(linreg, X, y, cv=5, scoring="neg_mean_squared_error"
cv_5_results
```

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

```
cv_5_results.mean()
```

```
0.17702834210001128
```

Compare and contrast the results. What is the difference between the train-test split and cross-validation results? Do you "trust" one more than the other?

```
"""
Rounded to 1 decimal place, both the train-test split result and the cross-validatic
result would be 0.2

However with more decimal places, the differences become apparent. The train-test sp
result is about 0.152 whereas the average cross-validation result is about 0.177. Be
this is an error-based metric, a higher value is worse, so this means that the trair
split result is "better" (more optimistic)

Another way to look at the results would be to compare the train-test split result t
of the 5 split results. Only 1 out of 5 splits has a better MSE than the train-test
meaning that the average is not being skewed by just 1 or 2 values that are signific
worse than the train-test split result

Taking the average as well as the spread into account, I would be more inclined to "
the cross-validated (less optimistic) score and assume that the performance on unsee
would be closer to 0.177 than 0.152
"""
```

Level Up: Let's Build It from Scratch!

Create a Cross-Validation Function

Write a function `kfolds(data, k)` 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.

For example, if you had this dataset:

```
example_data = pd.DataFrame({
    "color": ["red", "orange", "yellow", "green", "blue", "indigo", "violet"]
})
example_data
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	color
0	red
1	orange
2	yellow
3	green
4	blue
5	indigo
6	violet

`kfolds(example_data, 3)` should return:

- a dataframe with `red`, `orange`, `yellow`
- a dataframe with `green`, `blue`
- a dataframe with `indigo`, `violet`

Because the example dataframe has 7 records, which is not evenly divisible by 3, so the "leftover" 1 record extends the length of the first dataframe.

```python
def kfolds(data, k):
    folds = []

    # Calculate the fold size plus the remainder
    num_observations = len(data)
    small_fold_size = num_observations // k
    large_fold_size = small_fold_size + 1
    leftovers = num_observations % k

    start_index = 0
    for fold_n in range(k):
        # Select fold size based on whether or not all of the leftovers have been us
        if fold_n < leftovers:
            fold_size = large_fold_size
        else:
            fold_size = small_fold_size

        # Get fold from dataframe and add it to the list
        fold = data.iloc[start_index:start_index + fold_size]
        folds.append(fold)

        # Move the start index to the start of the next fold
        start_index += fold_size

    return folds

results = kfolds(example_data, 3)
for result in results:
    print(result, "\n")
```

```
    color
0     red
1  orange
2  yellow

   color
3  green
4   blue

    color
5  indigo
6  violet
```

Apply Your Function to the Ames Housing Data

Get folds for both `X` and `y`.

```
X_folds = kfolds(X, 5)
y_folds = kfolds(y, 5)
```

Perform a Linear Regression for Each Fold and Calculate the Test Error

Remember that for each fold you will need to concatenate all but one of the folds to represent the training data, while the one remaining fold represents the test data.

```
test_errs = []
k = 5

for n in range(k):
    # Split into train and test for the fold
    X_train = pd.concat([fold for i, fold in enumerate(X_folds) if i!=n])
    X_test = X_folds[n]
    y_train = pd.concat([fold for i, fold in enumerate(y_folds) if i!=n])
    y_test = y_folds[n]

    # Fit a linear regression model
    linreg.fit(X_train, y_train)

    # Evaluate test errors
    y_hat_test = linreg.predict(X_test)
    test_residuals = y_hat_test - y_test
    test_errs.append(np.mean(test_residuals.astype(float)**2))

print(test_errs)
```

```
[0.12431546148437427, 0.19350064631313135, 0.18910530431311193,
0.17079325250026922, 0.20742704588916958]
```

If your code was written correctly, these should be the same errors as scikit-learn produced with `cross_val_score` (within rounding error). Test this out below:

```
for k in range(5):
    print(f"Split {k+1}")
    print(f"My result:      {round(test_errs[k], 4)}")
    print(f"sklearn result: {round(cv_5_results[k], 4)}\n")
```

```
Split 1
My result:      0.1243
sklearn result: 0.1243

Split 2
My result:      0.1935
sklearn result: 0.1935

Split 3
My result:      0.1891
sklearn result: 0.1891

Split 4
My result:      0.1708
sklearn result: 0.1708

Split 5
My result:      0.2074
sklearn result: 0.2074
```

This was a bit of work! Hopefully you have a clearer understanding of the underlying logic for cross-validation if you attempted this exercise.

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 more robust than train-test split.

Releases

No releases published

Packages

No packages published

Contributors 4

LoreDirick Lore Dirick

hoffm386 Erin R Hoffman

sumedh10 Sumedh Panchadhar

mas16 matt

Languages

- **Jupyter Notebook** 100.0%