index

January 26, 2022

1 Regression Model Validation - Lab

1.1 Introduction

In this lab, you'll be able to validate your Ames Housing data model using train-test split.

1.2 Objectives

You will be able to:

- Compare training and testing errors to determine if model is over or underfitting

1.3 Let's use our Ames Housing Data again!

We included the code to preprocess below.

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

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

# using 9 predictive categorical or continuous features,
# plus the target SalePrice

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)
```

[3]:
```
X = preprocessed.drop('SalePrice_log', axis=1)
y = preprocessed['SalePrice_log']
```

1.3.1 Perform a train-test split

[7]:
```
# Split the data into training and test sets. Use the default split size
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_state=42)
```

1.3.2 Apply your model to the train set

[13]:
```
# Import and initialize the linear regression model class
from sklearn.linear_model import LinearRegression
```

[14]:
```
# Fit the model to train data
model = LinearRegression()
model.fit(X_train, y_train)
```

[14]:
```
LinearRegression()
```

1.3.3 Calculate predictions on training and test sets

[33]:
```
# Calculate predictions on training and test sets

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

# y_test_pred_I = y_test_pred.flatten()
# y_train_pred_I = y_train_pred.flatten()
```

1.3.4 Calculate training and test residuals

[41]:
```python
# Calculate residuals
train_res = y_train_pred - y_train
test_res = y_test_pred - y_test

MSE_train = np.inner(train_res,train_res) / len(train_res)
MSE_test = np.inner(test_res,test_res) / len(test_res)

print('Train Mean Squarred Error:', MSE_train)
print('Test Mean Squarred Error:', MSE_test)
```

```
Train Mean Squarred Error: 0.16025695964655187
Test Mean Squarred Error: 0.17595331097301373
```

1.3.5 Calculate the Mean Squared Error (MSE)

A good way to compare overall performance is to compare the mean squarred error for the predicted values on the training and test sets.

[42]:
```python
# Import mean_squared_error from sklearn.metrics
from sklearn.metrics import mean_squared_error
```

[43]:
```python
# Calculate training and test MSE
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: 0.16025695964655184
Test Mean Squarred Error: 0.17595331097301375
```

If your test error is substantially worse than the train error, this is a sign that the model doesn't generalize well to future cases.

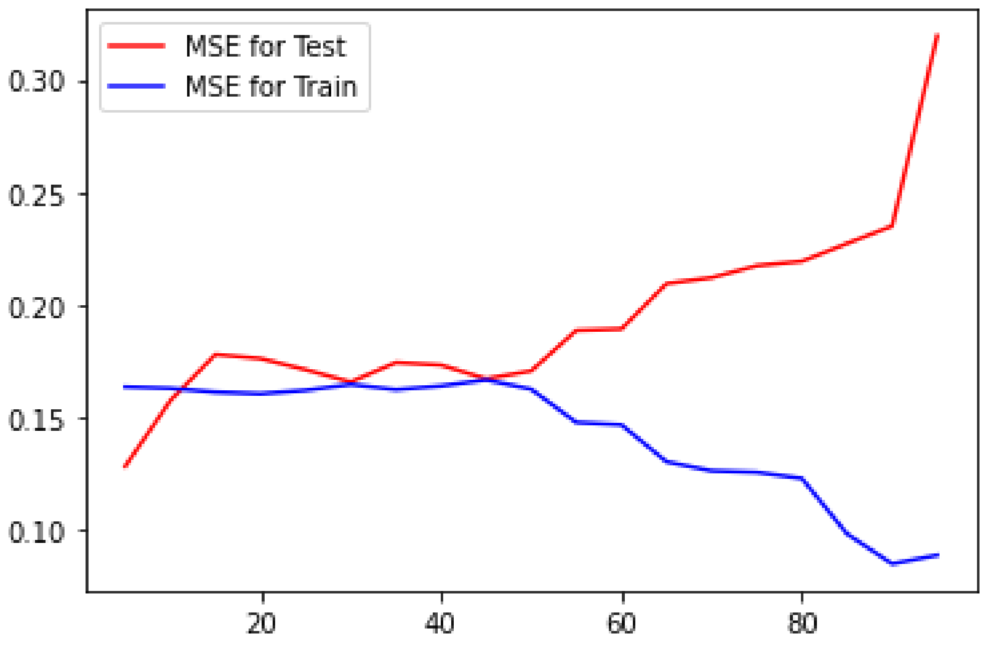
One simple way to demonstrate overfitting and underfitting is to alter the size of our train-test split. By default, scikit-learn allocates 25% of the data to the test set and 75% to the training set. Fitting a model on only 10% of the data is apt to lead to underfitting, while training a model on 99% of the data is apt to lead to overfitting.

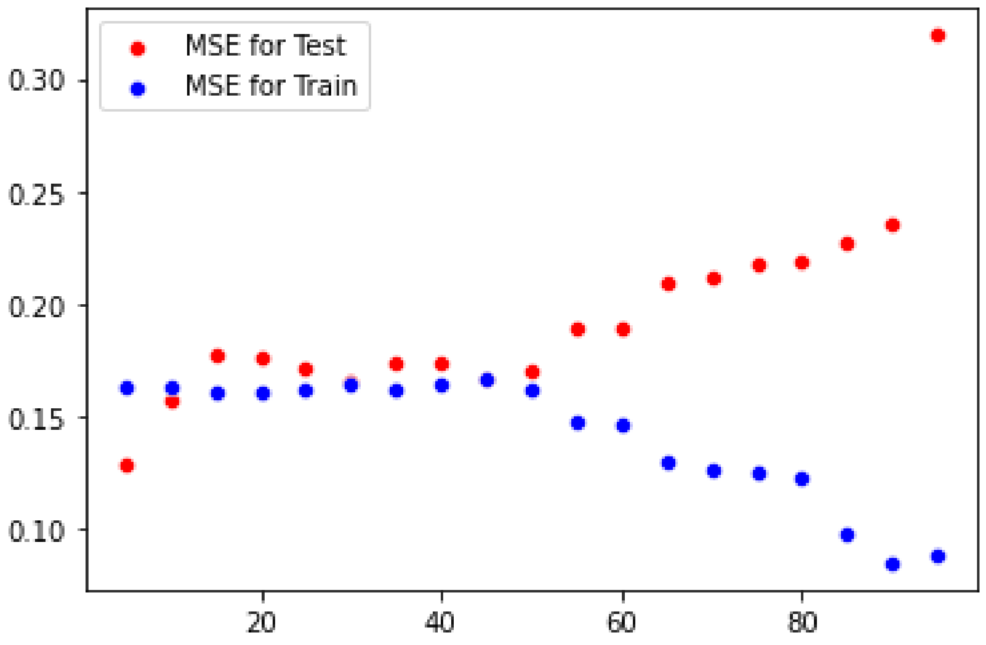
2 Evaluate the effect of train-test split size

Iterate over a range of train-test split sizes from .5 to .95. For each of these, generate a new train/test split sample. Fit a model to the training sample and calculate both the training error and the test error (mse) for each of these splits. Plot these two curves (train error vs. training size and test error vs. training size) on a graph.

[60]:
```python
# Your code here
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# import random
# random.seed(110)

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

MSE_test = []
MSE_train = []
split = range(5, 100, 5)


for item in split:
    X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        test_size=item/100,
                                                        random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)

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

    train_mse = mean_squared_error(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)

    MSE_test.append(test_mse)
    MSE_train.append(train_mse)
```

[61]:
```
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline

sns.lineplot(x = split, y = MSE_test, color = "red", label = "MSE for Test");
sns.lineplot(x = split, y = MSE_train, color = "blue", label = "MSE for Train");
plt.show()

sns.scatterplot(x = split, y = MSE_test, color = "red", label = "MSE for Test");
sns.scatterplot(x = split, y = MSE_train, color = "blue", label = "MSE for␣
 ↪Train");
plt.show()
```





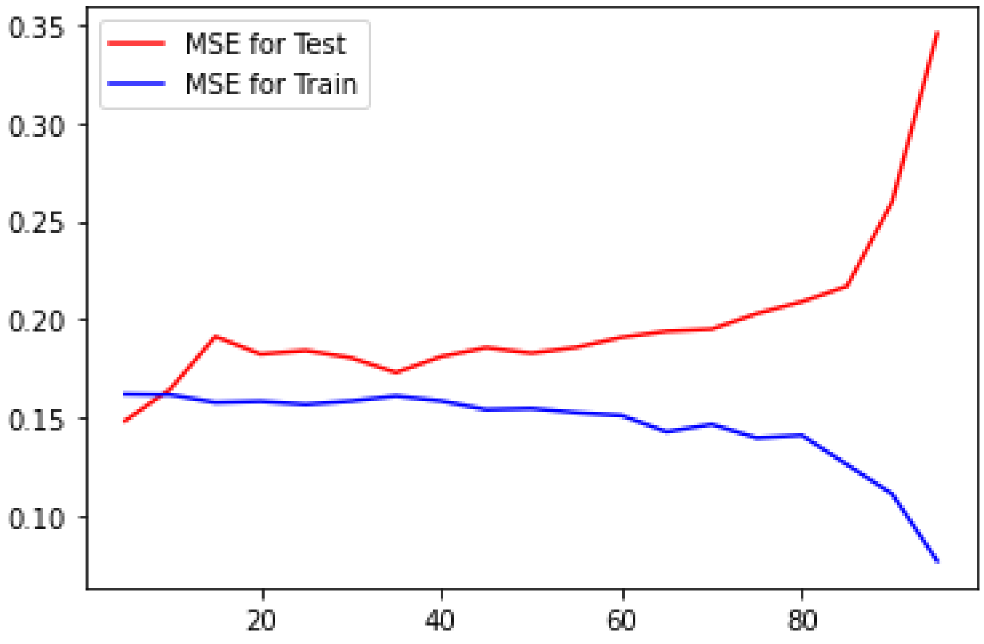
3 Evaluate the effect of train-test split size: Extension

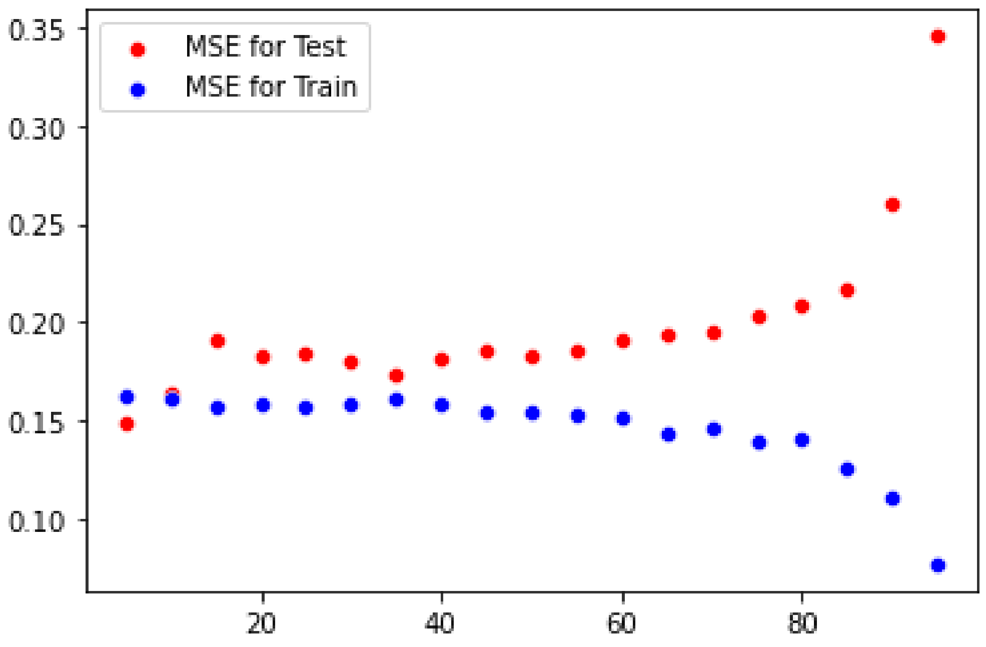
Repeat the previous example, but for each train-test split size, generate 10 iterations of models/errors and save the average train/test error. This will help account for any particularly good/bad models that might have resulted from poor/good splits in the data.

[70]:
```
# Your code here

# Your code here
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# import random
# random.seed(900)

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

split = range(5, 100, 5)

MSE_test_II = []
MSE_train_II = []
for item in split:
    MSE_test_err = []
    MSE_train_err = []

    for i in range(10):
        X_train, X_test, y_train, y_test = train_test_split(X,
                                                            y,
                                                            test_size=item/100)

        model = LinearRegression()
        model.fit(X_train, y_train)

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

        train_mse = mean_squared_error(y_train, y_train_pred)
        test_mse = mean_squared_error(y_test, y_test_pred)

        MSE_test_err.append(test_mse)
        MSE_train_err.append(train_mse)

    MSE_test_II.append(np.mean(MSE_test_err))
    MSE_train_II.append(np.mean(MSE_train_err))
```

[71]:
```
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
```

```
sns.lineplot(x = split, y = MSE_test_II, color = "red", label = "MSE for Test");
sns.lineplot(x = split, y = MSE_train_II, color = "blue", label = "MSE for␣
↪Train");
plt.show()

sns.scatterplot(x = split, y = MSE_test_II, color = "red", label = "MSE for␣
↪Test");
sns.scatterplot(x = split, y = MSE_train_II, color = "blue", label = "MSE for␣
↪Train");
plt.show()
```





What's happening here? Evaluate your result!

3.1 Summary

Congratulations! You now practiced your knowledge of MSE and used your train-test split skills to validate your model.