model-evaluation-and-refinement

February 22, 2022

1 Model Evaluation and Refinement

Estimated time needed: **30** minutes

1.1 Objectives

After completing this lab you will be able to:

- Evaluate and refine prediction models

Table of Contents

Model Evaluation

Over-fitting, Under-fitting and Model Selection

Ridge Regression

Grid Search

This dataset was hosted on IBM Cloud object. Click HERE for free storage.

[ ]:
```
#install specific version of libraries used in lab
#! mamba install pandas==1.3.3 -y
#! mamba install numpy=1.21.2 -y
#! mamba install sklearn=0.20.1 -y
#! mamba install   ipywidgets=7.4.2 -y
```

[1]:
```
import pandas as pd
import numpy as np

# Import clean data
path = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↪IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/
↪module_5_auto.csv'
df = pd.read_csv(path)
```

[2]:
```
df.to_csv('module_5_auto.csv')
```

First, let's only use numeric data:

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

[3]:
```
   Unnamed: 0  Unnamed: 0.1  symboling  normalized-losses  wheel-base  \
0           0             0          3                122        88.6
1           1             1          3                122        88.6
2           2             2          1                122        94.5
3           3             3          2                164        99.8
4           4             4          2                164        99.4

     length     width  height  curb-weight  engine-size  …  stroke  \
0  0.811148  0.890278    48.8         2548          130  …    2.68
1  0.811148  0.890278    48.8         2548          130  …    2.68
2  0.822681  0.909722    52.4         2823          152  …    3.47
3  0.848630  0.919444    54.3         2337          109  …    3.40
4  0.848630  0.922222    54.3         2824          136  …    3.40

   compression-ratio  horsepower  peak-rpm  city-mpg  highway-mpg    price  \
0                9.0       111.0    5000.0        21           27  13495.0
1                9.0       111.0    5000.0        21           27  16500.0
2                9.0       154.0    5000.0        19           26  16500.0
3               10.0       102.0    5500.0        24           30  13950.0
4                8.0       115.0    5500.0        18           22  17450.0

   city-L/100km  diesel  gas
0     11.190476       0    1
1     11.190476       0    1
2     12.368421       0    1
3      9.791667       0    1
4     13.055556       0    1

[5 rows x 21 columns]
```

Libraries for plotting:

[5]:
```
from ipywidgets import interact, interactive, fixed, interact_manual
```

Functions for Plotting

[6]:
```
def DistributionPlot(RedFunction, BlueFunction, RedName, BlueName, Title):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))

    ax1 = sns.distplot(RedFunction, hist=False, color="r", label=RedName)
    ax2 = sns.distplot(BlueFunction, hist=False, color="b", label=BlueName,␣
 ↪ax=ax1)
```

```
    plt.title(Title)
    plt.xlabel('Price (in dollars)')
    plt.ylabel('Proportion of Cars')

    plt.show()
    plt.close()
```

[8]:
```
def PollyPlot(xtrain, xtest, y_train, y_test, lr,poly_transform):
    width = 12
    height = 10
    plt.figure(figsize=(width, height))


    #training data
    #testing data
    # lr:  linear regression object
    #poly_transform:  polynomial transformation object

    xmax=max([xtrain.values.max(), xtest.values.max()])

    xmin=min([xtrain.values.min(), xtest.values.min()])

    x=np.arange(xmin, xmax, 0.1)


    plt.plot(xtrain, y_train, 'ro', label='Training Data')
    plt.plot(xtest, y_test, 'go', label='Test Data')
    plt.plot(x, lr.predict(poly_transform.fit_transform(x.reshape(-1, 1))),␣
 ↪label='Predicted Function')
    plt.ylim([-10000, 60000])
    plt.ylabel('Price')
    plt.legend()
```

Part 1: Training and Testing

An important step in testing your model is to split your data into training and testing data. We will place the target data price in a separate dataframe y_data:

[9]:
```
y_data = df['price']
```

Drop price data in dataframe **x_data**:

[11]:
```
x_data=df.drop('price',axis=1)
```

Now, we randomly split our data into training and testing data using the function train_test_split.

[12]:
```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.
↪10, random_state=1)


print("number of test samples :", x_test.shape[0])
print("number of training samples:",x_train.shape[0])
```

```
number of test samples : 21
number of training samples: 180
```

The test_size parameter sets the proportion of data that is split into the testing set. In the above, the testing set is 10% of the total dataset.

Question #1):

Use the function "train_test_split" to split up the dataset such that 40% of the data samples will be utilized for testing. Set the parameter "random_state" equal to zero. The output of the function should be the following: "x_train1" , "x_test1", "y_train1" and "y_test1".

[15]:
```
# Write your code below and press Shift+Enter to execute
x_train1, x_test1, y_train1, y_test1 = train_test_split(x_data, y_data,␣
↪test_size=0.4, random_state=0)
print("number of test samples :", x_test1.shape[0])
print("number of training samples:",x_train1.shape[0])
```

```
number of test samples : 81
number of training samples: 120
```

Click here for the solution

```
x_train1, x_test1, y_train1, y_test1 = train_test_split(x_data, y_data, test_size=0.4, random_s
print("number of test samples :", x_test1.shape[0])
print("number of training samples:",x_train1.shape[0])
```

Let's import LinearRegression from the module linear_model.

[18]:
```
from sklearn.linear_model import LinearRegression
```

We create a Linear Regression object:

[19]:
```
lre=LinearRegression()
```

We fit the model using the feature "horsepower":

[20]:
```
lre.fit(x_train[['horsepower']], y_train)
```

[20]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Let's calculate the R^2 on the test data:

[21]:
```python
lre.score(x_test[['horsepower']], y_test)
```

[21]: 0.3635875575078824

We can see the R^2 is much smaller using the test data compared to the training data.

[22]:
```python
lre.score(x_train[['horsepower']], y_train)
```

[22]: 0.6619724197515103

Question #2):

Find the R^2 on the test data using 40% of the dataset for testing.

[27]:
```python
# Write your code below and press Shift+Enter to execute
x_train1, x_test1, y_train1, y_test1 = train_test_split(x_data, y_data, test_size=0.4, random_state=0)
lre.fit(x_train1[['horsepower']],y_train1)
lre.score(x_test1[['horsepower']],y_test1)
```

[27]: 0.7139364665406973

Click here for the solution

```python
x_train1, x_test1, y_train1, y_test1 = train_test_split(x_data, y_data, test_size=0.4, random_s
lre.fit(x_train1[['horsepower']],y_train1)
lre.score(x_test1[['horsepower']],y_test1)
```

Sometimes you do not have sufficient testing data; as a result, you may want to perform cross-validation. Let's go over several methods that you can use for cross-validation.

Cross-Validation Score

Let's import model_selection from the module cross_val_score.

[28]:
```python
from sklearn.model_selection import cross_val_score
```

We input the object, the feature ("horsepower"), and the target data (y_data). The parameter 'cv' determines the number of folds. In this case, it is 4.

[30]:
```python
Rcross = cross_val_score(lre, x_data[['horsepower']], y_data, cv=4)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:437: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
```

```
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
```

The default scoring is R^2. Each element in the array has the average R^2 value for the fold:

[31]:
```
Rcross
```

[31]:
```
array([0.7746232 , 0.51716687, 0.74785353, 0.04839605])
```

We can calculate the average and standard deviation of our estimate:

[32]:
```
print("The mean of the folds are", Rcross.mean(), "and the standard deviation␣
↪is" , Rcross.std())
```

```
The mean of the folds are 0.522009915042119 and the standard deviation is
0.2911839444756029
```

We can use negative squared error as a score by setting the parameter 'scoring' metric to 'neg_mean_squared_error'.

[33]:
```
-1 * cross_val_score(lre,x_data[['horsepower']],␣
↪y_data,cv=4,scoring='neg_mean_squared_error')
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:437: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
```

[33]: array([20254142.84026704, 43745493.2650517 , 12539630.34014931,
        17561927.72247591])

Question #3):

Calculate the average R^2 using two folds, then find the average R^2 for the second fold utilizing the "horsepower" feature:

[34]:
```
# Write your code below and press Shift+Enter to execute
Rc=cross_val_score(lre,x_data[['horsepower']], y_data,cv=2)
Rc.mean()
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:437: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
```

[34]: 0.5166761697127429

Click here for the solution

```
Rc=cross_val_score(lre,x_data[['horsepower']], y_data,cv=2)
Rc.mean()
```

You can also use the function 'cross_val_predict' to predict the output. The function splits up the data into the specified number of folds, with one fold for testing and the other folds are used for training. First, import the function:

[35]:
```python
from sklearn.model_selection import cross_val_predict
```

We input the object, the feature "horsepower", and the target data y_data. The parameter 'cv' determines the number of folds. In this case, it is 4. We can produce an output:

[36]:
```python
yhat = cross_val_predict(lre,x_data[['horsepower']], y_data,cv=4)
yhat[0:5]
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:437: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
```

```
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
```

[36]:
```
array([14141.63807508, 14141.63807508, 20814.29423473, 12745.03562306,
       14762.35027598])
```

Part 2: Overfitting, Underfitting and Model Selection

It turns out that the test data, sometimes referred to as the "out of sample data", is a much better measure of how well your model performs in the real world. One reason for this is overfitting.

Let's go over some examples. It turns out these differences are more apparent in Multiple Linear Regression and Polynomial Regression so we will explore overfitting in that context.

Let's create Multiple Linear Regression objects and train the model using 'horsepower', 'curb-weight', 'engine-size' and 'highway-mpg' as features.

[37]:
```
lr = LinearRegression()
lr.fit(x_train[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],␣
 ↪y_train)
```

[37]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Prediction using training data:

[38]:
```
yhat_train = lr.predict(x_train[['horsepower', 'curb-weight', 'engine-size',␣
 ↪'highway-mpg']])
yhat_train[0:5]
```

[38]:
```
array([ 7426.6731551 , 28323.75090803, 14213.38819709,  4052.34146983,
       34500.19124244])
```

Prediction using test data:

[39]:
```
yhat_test = lr.predict(x_test[['horsepower', 'curb-weight', 'engine-size',␣
 ↪'highway-mpg']])
yhat_test[0:5]
```

[39]:
```
array([11349.35089149,  5884.11059106, 11208.6928275 ,  6641.07786278,
       15565.79920282])
```

Let's perform some model evaluation using our training and testing data separately. First, we import the seaborn and matplotlib library for plotting.

[40]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Let's examine the distribution of the predicted values of the training data.

```
[43]: Title = 'Distribution  Plot of  Predicted Value Using Training Data vs Training␣
       ↪Data Distribution'
      DistributionPlot(y_train, yhat_train, "Actual Values (Train)", "Predicted␣
       ↪Values (Train)", Title)
```

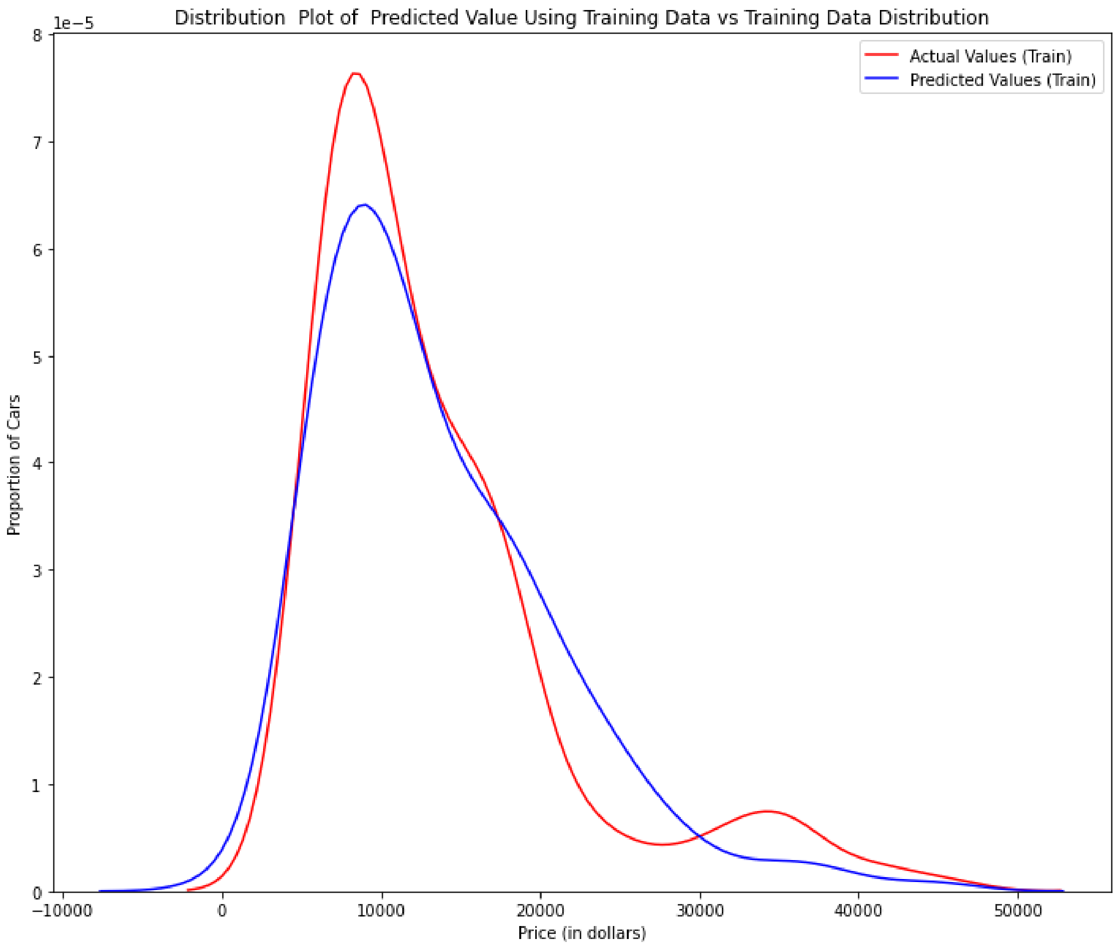


Figure 1: Plot of predicted values using the training data compared to the actual values of the training data.

So far, the model seems to be doing well in learning from the training dataset. But what happens when the model encounters new data from the testing dataset? When the model generates new values from the test data, we see the distribution of the predicted values is much different from the actual target values.

```
[44]: Title='Distribution  Plot of  Predicted Value Using Test Data vs Data␣
       ↪Distribution of Test Data'
      DistributionPlot(y_test,yhat_test,"Actual Values (Test)","Predicted Values␣
       ↪(Test)",Title)
```

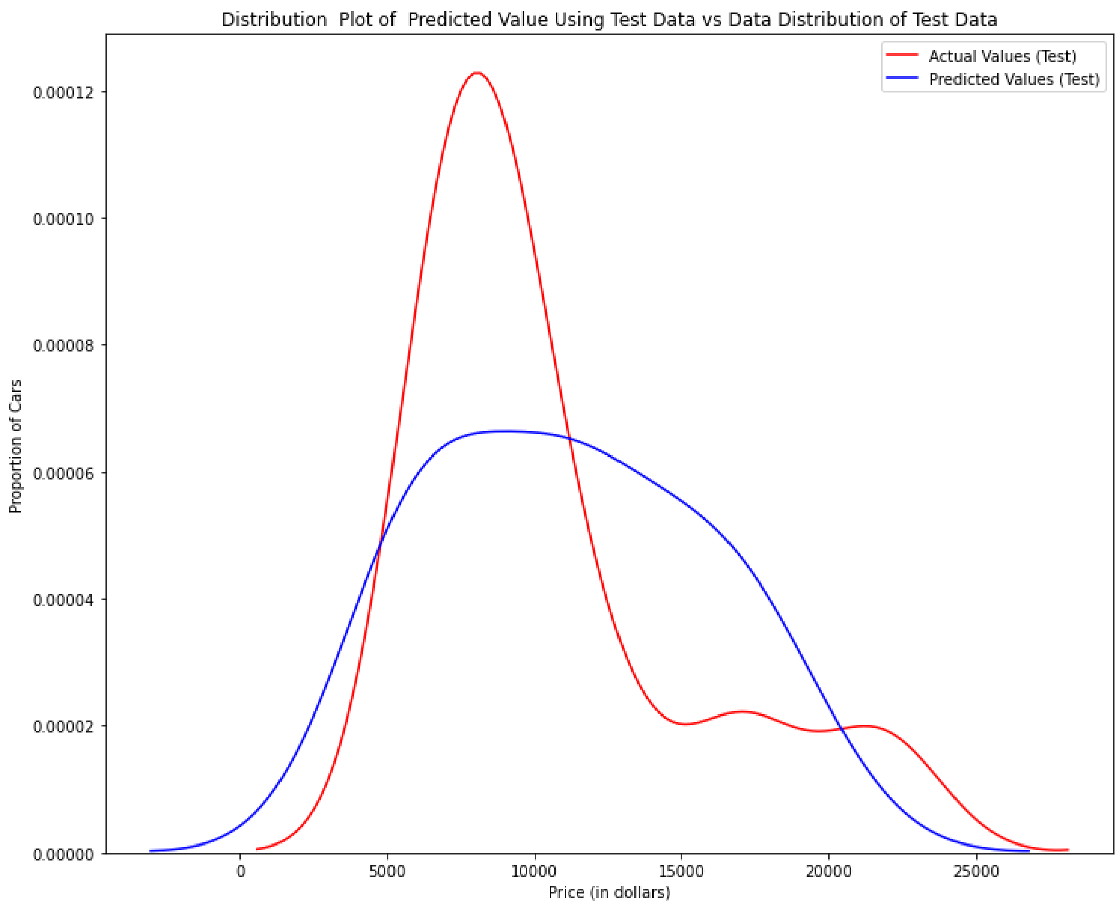


Figure 2: Plot of predicted value using the test data compared to the actual values of the test data.

Comparing Figure 1 and Figure 2, it is evident that the distribution of the test data in Figure 1 is much better at fitting the data. This difference in Figure 2 is apparent in the range of 5000 to 15,000. This is where the shape of the distribution is extremely different. Let's see if polynomial regression also exhibits a drop in the prediction accuracy when analysing the test dataset.

[46]:
```
from sklearn.preprocessing import PolynomialFeatures
```

Overfitting

Overfitting occurs when the model fits the noise, but not the underlying process. Therefore, when testing your model using the test set, your model does not perform as well since it is modelling noise, not the underlying process that generated the relationship. Let's create a degree 5 polynomial model.

Let's use 55 percent of the data for training and the rest for testing:

[49]:
```
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.
↪45, random_state=0)
```

We will perform a degree 5 polynomial transformation on the feature 'horsepower'.

[50]:
```
pr = PolynomialFeatures(degree=5)
x_train_pr = pr.fit_transform(x_train[['horsepower']])
x_test_pr = pr.fit_transform(x_test[['horsepower']])
pr
```

[50]:
```
PolynomialFeatures(degree=5, include_bias=True, interaction_only=False)
```

Now, let's create a Linear Regression model "poly" and train it.

[52]:
```
poly = LinearRegression()
poly.fit(x_train_pr, y_train)
```

[52]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

We can see the output of our model using the method "predict." We assign the values to "yhat".

[53]:
```
yhat = poly.predict(x_test_pr)
yhat[0:5]
```

[53]:
```
array([ 6728.73285076,  7308.05589332, 12213.80614303, 18893.13997531,
       19995.82734265])
```

Let's take the first five predicted values and compare it to the actual targets.

[54]:
```
print("Predicted values:", yhat[0:4])
print("True values:", y_test[0:4].values)
```

```
Predicted values: [ 6728.73285076  7308.05589332 12213.80614303 18893.13997531]
True values: [ 6295. 10698. 13860. 13499.]
```

We will use the function "PollyPlot" that we defined at the beginning of the lab to display the training data, testing data, and the predicted function.

[55]:
```
PollyPlot(x_train[['horsepower']], x_test[['horsepower']], y_train, y_test,␣
 ↪poly,pr)
```

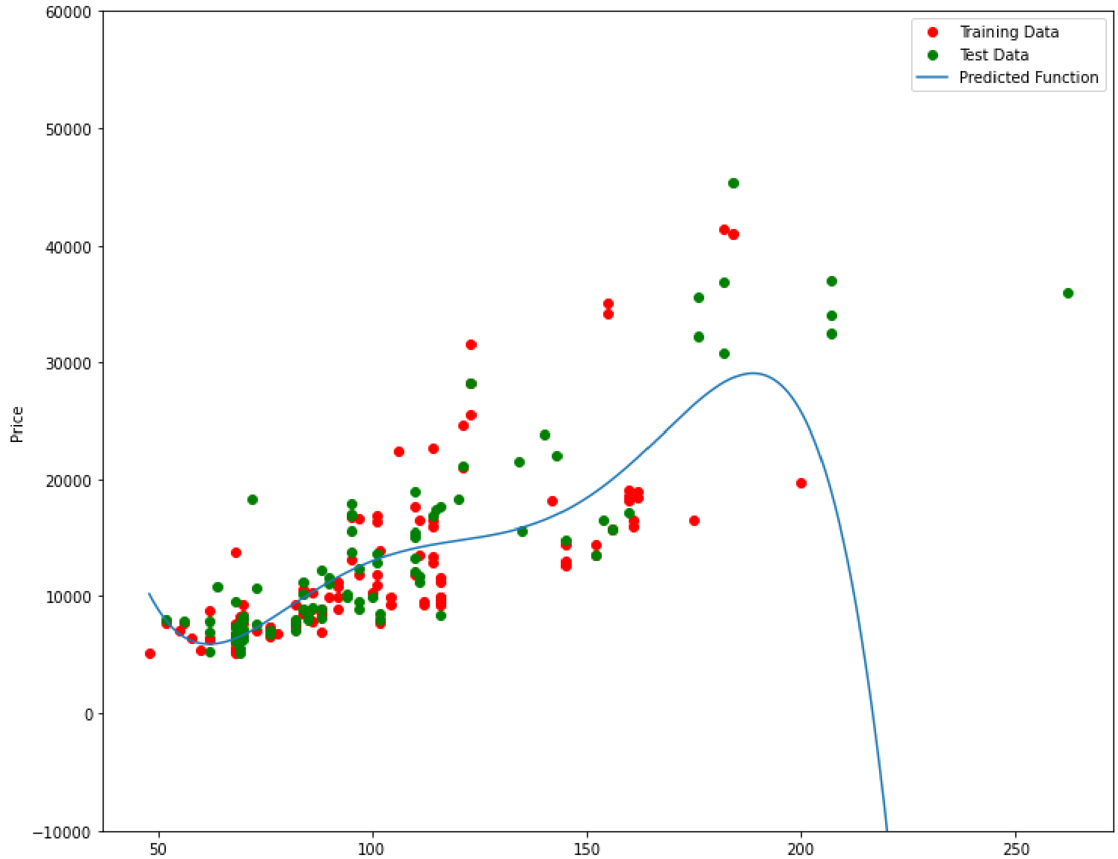


Figure 3: A polynomial regression model where red dots represent training data, green dots represent test data, and the blue line represents the model prediction.

We see that the estimated function appears to track the data but around 200 horsepower, the function begins to diverge from the data points.

R^2 of the training data:

[56]:
```
poly.score(x_train_pr, y_train)
```

[56]:
```
0.5567716902237296
```

R^2 of the test data:

[57]:
```
poly.score(x_test_pr, y_test)
```

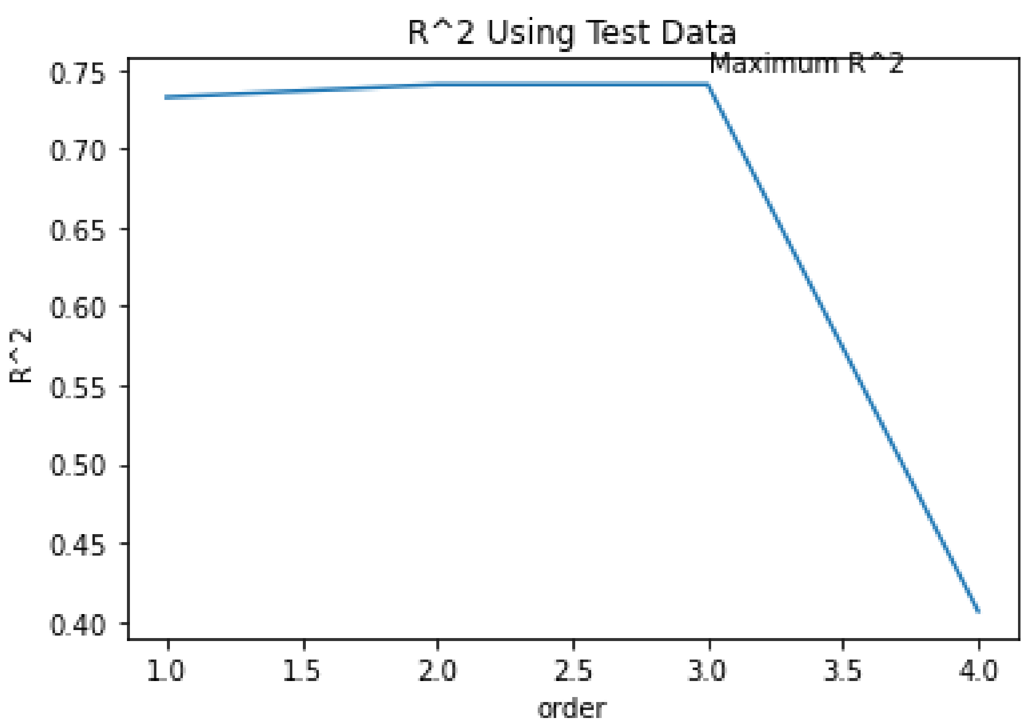
[57]:
```
-29.87158580724305
```

We see the R^2 for the training data is 0.5567 while the R^2 on the test data was -29.87. The lower the R^2, the worse the model. A negative R^2 is a sign of overfitting.

Let's see how the R^2 changes on the test data for different order polynomials and then plot the results:

[58]:
```python
Rsqu_test = []

order = [1, 2, 3, 4]
for n in order:
    pr = PolynomialFeatures(degree=n)

    x_train_pr = pr.fit_transform(x_train[['horsepower']])

    x_test_pr = pr.fit_transform(x_test[['horsepower']])

    lr.fit(x_train_pr, y_train)

    Rsqu_test.append(lr.score(x_test_pr, y_test))

plt.plot(order, Rsqu_test)
plt.xlabel('order')
plt.ylabel('R^2')
plt.title('R^2 Using Test Data')
plt.text(3, 0.75, 'Maximum R^2 ')
```

[58]:
```
Text(3, 0.75, 'Maximum R^2 ')
```



We see the R^2 gradually increases until an order three polynomial is used. Then, the R^2 dramatically decreases at an order four polynomial.

The following function will be used in the next section. Please run the cell below.

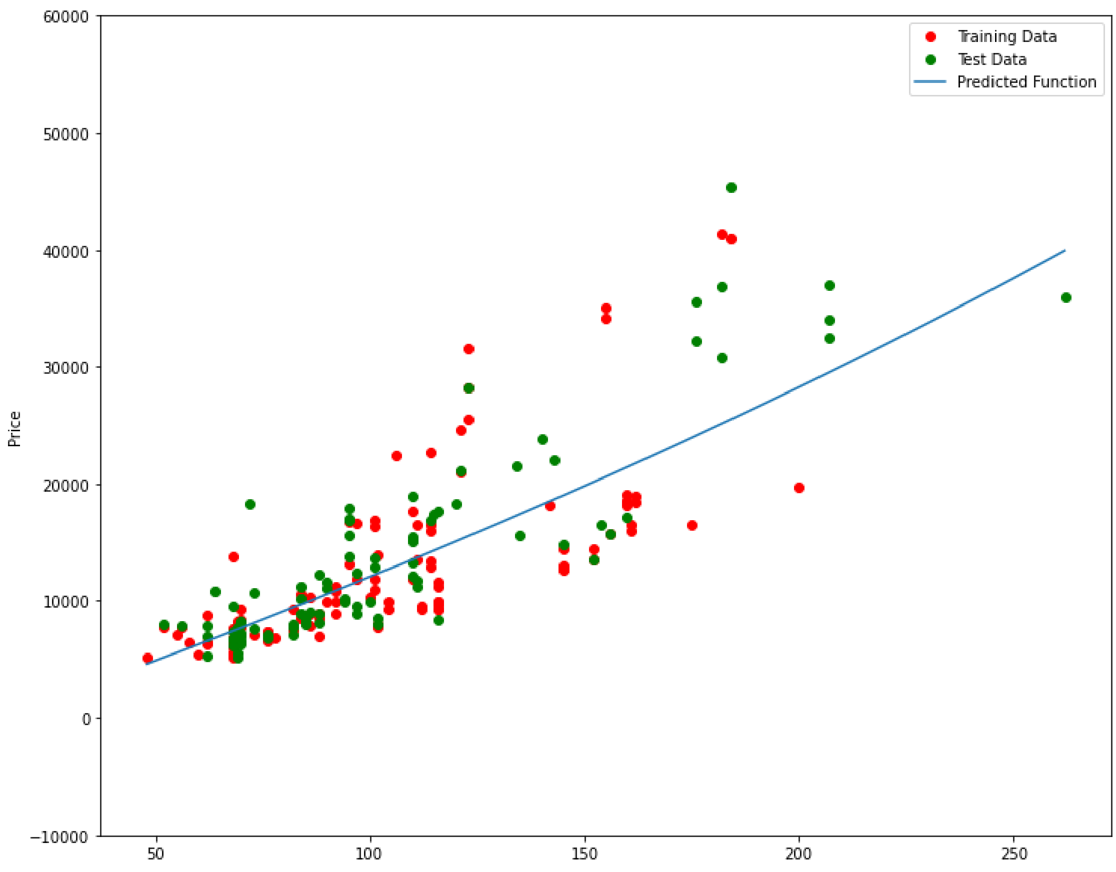
[59]:
```
def f(order, test_data):
    x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=test_data, random_state=0)
    pr = PolynomialFeatures(degree=order)
    x_train_pr = pr.fit_transform(x_train[['horsepower']])
    x_test_pr = pr.fit_transform(x_test[['horsepower']])
    poly = LinearRegression()
    poly.fit(x_train_pr,y_train)
    PollyPlot(x_train[['horsepower']], x_test[['horsepower']], y_train,y_test, poly, pr)
```

The following interface allows you to experiment with different polynomial orders and different amounts of data.

[60]:
```
interact(f, order=(0, 6, 1), test_data=(0.05, 0.95, 0.05))
```

```
interactive(children=(IntSlider(value=3, description='order', max=6), FloatSlider(value=0.45, d
```

[60]:
```
<function __main__.f(order, test_data)>
```



Question #4a):

We can perform polynomial transformations with more than one feature. Create a "PolynomialFeatures" object "pr1" of degree two.

[61]:
```
# Write your code below and press Shift+Enter to execute
pr1=PolynomialFeatures(degree=2)
```

Click here for the solution

```
pr1=PolynomialFeatures(degree=2)
```

Question #4b):

Transform the training and testing samples for the features 'horsepower', 'curb-weight', 'engine-size' and 'highway-mpg'. Hint: use the method "fit_transform".

[62]:
```
# Write your code below and press Shift+Enter to execute
x_train_pr1=pr1.fit_transform(x_train[['horsepower', 'curb-weight',␣
↪'engine-size', 'highway-mpg']])

x_test_pr1=pr1.fit_transform(x_test[['horsepower', 'curb-weight',␣
↪'engine-size', 'highway-mpg']])
```

Click here for the solution

```
x_train_pr1=pr1.fit_transform(x_train[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg

x_test_pr1=pr1.fit_transform(x_test[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']
```

Question #4c):

How many dimensions does the new feature have? Hint: use the attribute "shape".

[63]:
```
# Write your code below and press Shift+Enter to execute
x_train_pr1.shape #there are now 15 features
```

[63]:
```
(110, 15)
```

Click here for the solution

```
x_train_pr1.shape #there are now 15 features
```

Question #4d):

Create a linear regression model "poly1". Train the object using the method "fit" using the polynomial features.

[65]:
```
# Write your code below and press Shift+Enter to execute
poly1=LinearRegression().fit(x_train_pr1,y_train)
```

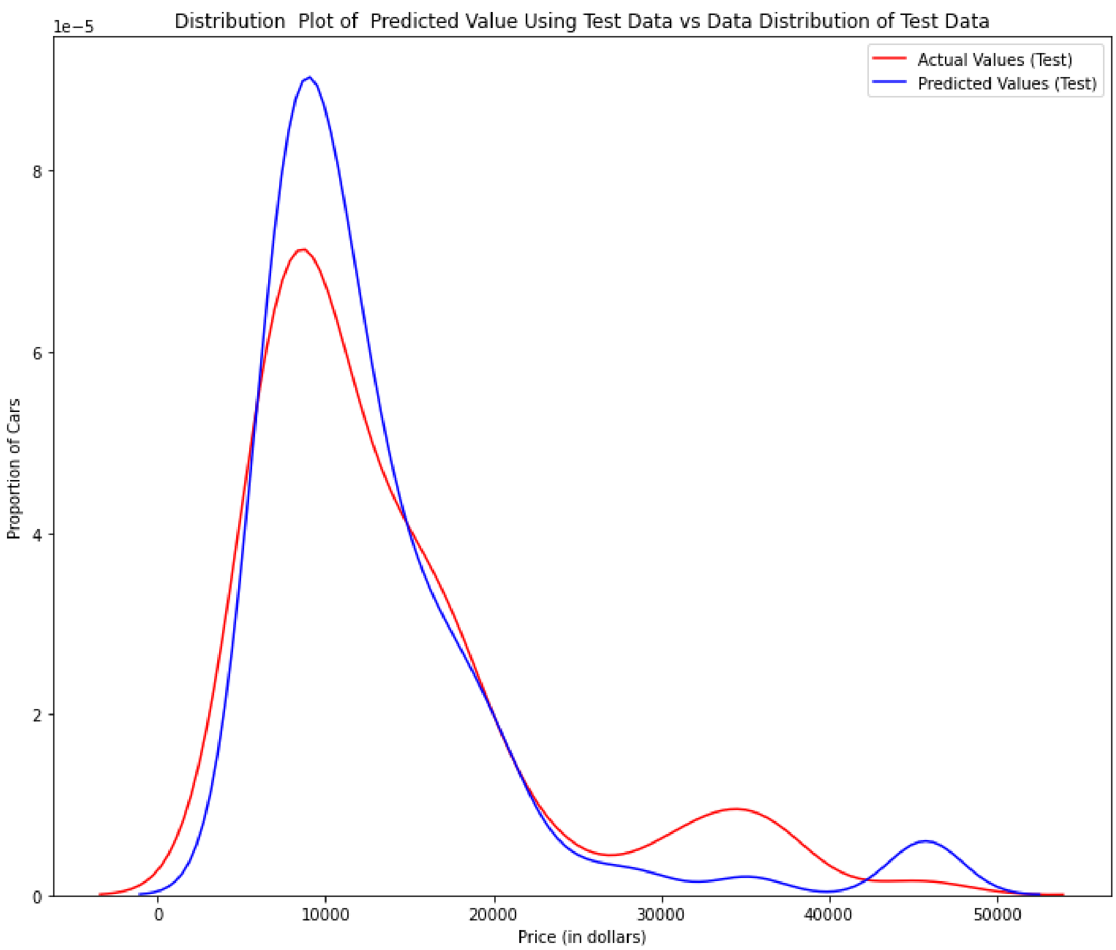
Click here for the solution

```
poly1=LinearRegression().fit(x_train_pr1,y_train)
```

Question #4e):

Use the method "predict" to predict an output on the polynomial features, then use the function "DistributionPlot" to display the distribution of the predicted test output vs. the actual test data.

```
[66]: # Write your code below and press Shift+Enter to execute
yhat_test1=poly1.predict(x_test_pr1)

Title='Distribution  Plot of  Predicted Value Using Test Data vs Data␣
 ↪Distribution of Test Data'

DistributionPlot(y_test, yhat_test1, "Actual Values (Test)", "Predicted Values␣
 ↪(Test)", Title)
```



Click here for the solution

```
yhat_test1=poly1.predict(x_test_pr1)

Title='Distribution  Plot of  Predicted Value Using Test Data vs Data Distribution of Test Data

DistributionPlot(y_test, yhat_test1, "Actual Values (Test)", "Predicted Values (Test)", Title)
```

Question #4f):

Using the distribution plot above, describe (in words) the two regions where the predicted prices are less accurate than the actual prices.

2 Write your code below and press Shift+Enter to execute

#The predicted value is higher than actual value for cars where the price $10,000 range, conversely the predicted price is lower than the price cost in the $30,000 to $40,000 range. As such the model is not as accurate in these ranges.

Click here for the solution

```
#The predicted value is higher than actual value for cars where the price $10,000 range, conve
```

Part 3: Ridge Regression

In this section, we will review Ridge Regression and see how the parameter alpha changes the model. Just a note, here our test data will be used as validation data.

Let's perform a degree two polynomial transformation on our data.

[67]:
```
pr=PolynomialFeatures(degree=2)
x_train_pr=pr.fit_transform(x_train[['horsepower', 'curb-weight',␣
↪'engine-size', 'highway-mpg','normalized-losses','symboling']])
x_test_pr=pr.fit_transform(x_test[['horsepower', 'curb-weight', 'engine-size',␣
↪'highway-mpg','normalized-losses','symboling']])
```

Let's import Ridge from the module linear models.

[68]:
```
from sklearn.linear_model import Ridge
```

Let's create a Ridge regression object, setting the regularization parameter (alpha) to 0.1

[69]:
```
RigeModel=Ridge(alpha=1)
```

Like regular regression, you can fit the model using the method fit.

[70]:
```
RigeModel.fit(x_train_pr, y_train)
```

[70]:
```
Ridge(alpha=1, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

[76]:
```
RigeModel.score(x_test_pr, y_test)
```

[76]: 0.5418576440207269

Similarly, you can obtain a prediction:

[71]:
```
yhat = RigeModel.predict(x_test_pr)
```

Let's compare the first five predicted samples to our test set:

[72]:
```
print('predicted:', yhat[0:4])
print('test set :', y_test[0:4].values)
```

```
predicted: [ 6570.82441941  9636.24891471 20949.92322737 19403.60313255]
test set : [ 6295. 10698. 13860. 13499.]
```

We select the value of alpha that minimizes the test error. To do so, we can use a for loop. We have also created a progress bar to see how many iterations we have completed so far.

[73]:
```
from tqdm import tqdm

Rsqu_test = []
Rsqu_train = []
dummy1 = []
Alpha = 10 * np.array(range(0,1000))
pbar = tqdm(Alpha)

for alpha in pbar:
    RigeModel = Ridge(alpha=alpha)
    RigeModel.fit(x_train_pr, y_train)
    test_score, train_score = RigeModel.score(x_test_pr, y_test), RigeModel.
 ↪score(x_train_pr, y_train)

    pbar.set_postfix({"Test Score": test_score, "Train Score": train_score})

    Rsqu_test.append(test_score)
    Rsqu_train.append(train_score)
```

```
100%|      | 1000/1000 [00:03<00:00, 250.27it/s, Test Score=0.564, Train
Score=0.859]
```

We can plot out the value of R^2 for different alphas:

[74]:
```
width = 12
height = 10
plt.figure(figsize=(width, height))

plt.plot(Alpha,Rsqu_test, label='validation data  ')
plt.plot(Alpha,Rsqu_train, 'r', label='training Data ')
plt.xlabel('alpha')
plt.ylabel('R^2')
```

```python
plt.legend()
```

[74]: <matplotlib.legend.Legend at 0x7fb760b224d0>

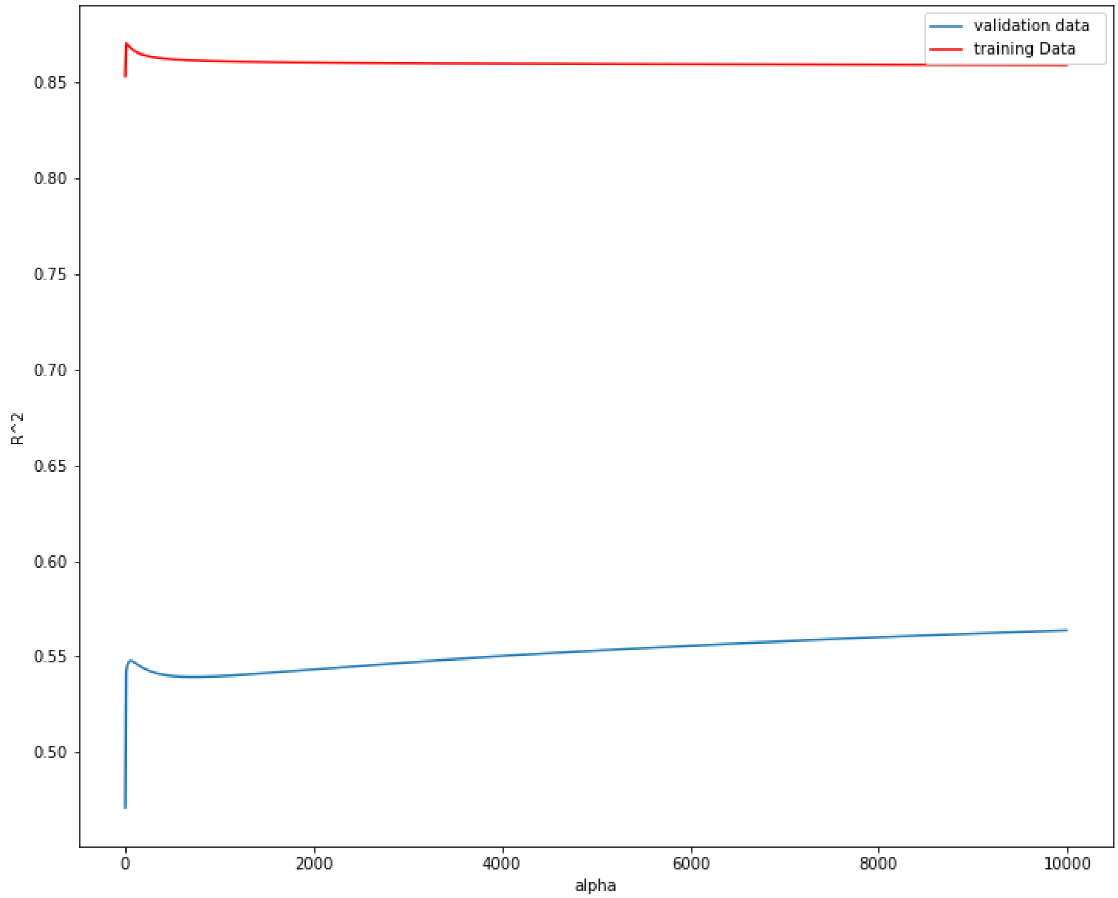


Figure 4: The blue line represents the R^2 of the validation data, and the red line represents the R^2 of the training data. The x-axis represents the different values of Alpha.

Here the model is built and tested on the same data, so the training and test data are the same.

The red line in Figure 4 represents the R^2 of the training data. As alpha increases the R^2 decreases. Therefore, as alpha increases, the model performs worse on the training data

The blue line represents the R^2 on the validation data. As the value for alpha increases, the R^2 increases and converges at a point.

Question #5):

Perform Ridge regression. Calculate the R^2 using the polynomial features, use the training data to train the model and use the test data to test the model. The parameter alpha should be set to 10.

[75]:
```
# Write your code below and press Shift+Enter to execute
RigeModel = Ridge(alpha=10)
RigeModel.fit(x_train_pr, y_train)
RigeModel.score(x_test_pr, y_test)
```

[75]: 0.5418576440207269

Click here for the solution

```
RigeModel = Ridge(alpha=10)
RigeModel.fit(x_train_pr, y_train)
RigeModel.score(x_test_pr, y_test)
```

Part 4: Grid Search

The term alpha is a hyperparameter. Sklearn has the class GridSearchCV to make the process of finding the best hyperparameter simpler.

Let's import GridSearchCV from the module model_selection.

[77]:
```
from sklearn.model_selection import GridSearchCV
```

We create a dictionary of parameter values:

[78]:
```
parameters1= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000, 100000, 100000]}]
parameters1
```

[78]: [{'alpha': [0.001, 0.1, 1, 10, 100, 1000, 10000, 100000, 100000]}]

Create a Ridge regression object:

[79]:
```
RR=Ridge()
RR
```

[79]:
```
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

Create a ridge grid search object:

[80]:
```
Grid1 = GridSearchCV(RR, parameters1,cv=4, iid=None)
```

In order to avoid a deprecation warning due to the iid parameter, we set the value of iid to "None".

Fit the model:

[81]:
```
Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],␣
↪y_data)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:437: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
```

```
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:821: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  dtype=np.int)
```

```
[81]: GridSearchCV(cv=4, error_score='raise-deprecating',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001),
       fit_params=None, iid=None, n_jobs=None,
       param_grid=[{'alpha': [0.001, 0.1, 1, 10, 100, 1000, 10000, 100000,
100000]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

The object finds the best parameter values on the validation data. We can obtain the estimator with the best parameters and assign it to the variable BestRR as follows:

```
[82]: BestRR=Grid1.best_estimator_
BestRR
```

```
[82]: Ridge(alpha=10000, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

We now test our model on the test data:

```
[83]: BestRR.score(x_test[['horsepower', 'curb-weight', 'engine-size',␣
 ↪'highway-mpg']], y_test)
```

```
[83]: 0.8411649831036152
```

Question #6):

Perform a grid search for the alpha parameter and the normalization parameter, then find the best values of the parameters:

```
[84]: # Write your code below and press Shift+Enter to execute
parameters2= [{'alpha': [0.001,0.1,1, 10, 100,␣
 ↪1000,10000,100000,100000],'normalize':[True,False]} ]
Grid2 = GridSearchCV(Ridge(), parameters2,cv=4)
Grid2.fit(x_data[['horsepower', 'curb-weight', 'engine-size',␣
 ↪'highway-mpg']],y_data)
Grid2.best_estimator_
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:437: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  fold_sizes = np.full(n_splits, n_samples // n_splits, dtype=np.int)
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:113: DeprecationWarning: `np.bool` is
a deprecated alias for the builtin `bool`. To silence this warning, use `bool`
by itself. Doing this will not modify any behavior and is safe. If you
specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  test_mask = np.zeros(_num_samples(X), dtype=np.bool)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:821: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by
itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the
precision. If you wish to review your current use, check the release note link
for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  dtype=np.int)
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:841: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

```
[84]: Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
         normalize=True, random_state=None, solver='auto', tol=0.001)
```

Click here for the solution

```
parameters2= [{'alpha': [0.001,0.1,1, 10, 100, 1000,10000,100000,100000],'normalize':[True,Fals
Grid2 = GridSearchCV(Ridge(), parameters2,cv=4)
Grid2.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']],y_data)
Grid2.best_estimator_
```

2.0.1 Thank you for completing this lab!

2.1 Author

Joseph Santarcangelo

2.1.1 Other Contributors

Mahdi Noorian PhD

Bahare Talayian

Eric Xiao

Steven Dong

Parizad

Hima Vasudevan

Fiorella Wenver

Yi Yao.

2.2 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-10-30	2.3	Lakshmi	Changed URL of csv
2020-10-05	2.2	Lakshmi	Removed unused library imports
2020-09-14	2.1	Lakshmi	Made changes in OverFitting section
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab

##