Model Evaluation and Refinement

* In sample evaluation tells us how well our model will fit the data used to train it.
* But it doesn’t inform us how well the trained model can be used to predict new data.
* Solution (Split them into two different categories)
  + sample data and training data.
  + Out of sample evaluation or test set
* Scikit learn package function (**train\_test\_split()**)
  + Code
    - **From sklearn.model\_selection import train\_test\_split**
    - **Text

      Description automatically generatedX\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_data,y\_data,test\_size=0.3, random\_state=0)**
* Cross Validation
  + Most common out of sample evaluation metrics
  + Chart, box and whisker chart

    Description automatically generatedMost effect use of data
  + Code (**Function cross\_val\_score()**)
    - **From sklearn.model\_selection import cross\_val\_score**
    - **Scores=cross\_val\_score(lr, x\_data, y\_data, cv = 3)** \*CV = Partitions
  + Code (**cross\_val\_predict()**) \*Returns a prediction obtained from each element.
    - **From sklearn.model\_selection import cross\_val\_predict**
    - **Yhat = cross\_val\_predict(lr2e, x\_data, y\_data, cv = 3)**
* Overfitting, Underfitting and Model Selection
  + Underfitting
    - When the model is too simple to fit the data.
    - Let's create Multiple Linear Regression objects and train the model using **'horsepower'**, **'curb-weight'**, **'engine-size'** and **'highway-mpg'** as features.
      * **lr = LinearRegression()**
      * **lr.fit(x\_train[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y\_train)**
    - Prediction using training data:
      * **yhat\_train = lr.predict(x\_train[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])**
      * **yhat\_train[0:5]**
    - Prediction using test data:
      * **yhat\_test = lr.predict(x\_test[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']])**
      * **yhat\_test[0:5]**
    - Let's perform some model evaluation using our training and testing data separately. First, we import the seaborn and matplotlib library for plotting.
      * **import matplotlib.pyplot as plt**
      * **%matplotlib inline**
      * **import seaborn as sns**
    - Let's examine the distribution of the predicted values of the training data
      * **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)**
      * **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)**
  + Overfitting
    - Is when the model is too flexible and fit the noise rather than the function.
      * Let's use 55 percent of the data for training and the rest for testing:
        + **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'**.
        + **pr = PolynomialFeatures(degree=5)**
        + **x\_train\_pr = pr.fit\_transform(x\_train[['horsepower']])**
        + **x\_test\_pr = pr.fit\_transform(x\_test[['horsepower']])**
        + **pr**
      * Now, let's create a Linear Regression model "poly" and train it.
        + **poly = LinearRegression()**
        + **poly.fit(x\_train\_pr, y\_train)**
      * We can see the output of our model using the method "predict." We assign the values to "yhat".
        + **yhat = poly.predict(x\_test\_pr)**
        + **yhat[0:5]**
      * Let's take the first five predicted values and compare it to the actual targets.
        + **print("Predicted values:", yhat[0:4])**
        + **print("True values:", y\_test[0:4].values)**
      * 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.
        + **PollyPlot(x\_train[['horsepower']], x\_test[['horsepower']], y\_train, y\_test, poly,pr)**
      * R^2 of the training data:
        + **poly.score(x\_train\_pr, y\_train)**
      * R^2 of the test data:
        + **poly.score(x\_test\_pr, y\_test)**
  + Now to calculate R^2 values in a (THIS WILL PLOT HOW R2 CHANGE WITH ORDER)
    - **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)**
      * **Rsque\_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 ')**
* Ridge Regression
  + Prevents overfitting. (If R^2 of 1 on Training Data and R^2 of 0 on Validation Data Increase alpha value)
  + Alpha a parameter is used to determine 0.1 to 10. If overfitting increase, if under fitting decrease.
  + Code:
    - **From sklearn.linear\_model import Ridge**
    - **RidgeModel=Ridge(alpha=0.1) \***Creating a ridge object
    - **RidgeModel.fit(x,y) \***Train the model
    - **Yhat = RidgeModel.predict(X) \***Make a prediction
  + Adjust alpha until R2 maximized or is closest to 1 as possible.
* Grid Search
  + Hyperparameters (Such as alpha in ridge regression)
    - Scikit-learn has a means of automatically iterating over these hyperparameters using cross-validation called grid search.
    - You use the validation data to test Hyperparameters.
  + Need to introduce parameters as a list
    - Ex: **parameters = [{‘alpha’ : [1, 10, 100, 1000] } ]**
      * \*Free parameter in this case Alpha is the key.
  + Example Code
    - **From sklearn.linear\_model import Ridge**
    - **From sklearn.model\_selection import GridSearchCV**
    - **Parameters1 = [{‘alpha’ : [0.001, 0.1, 1, 10, 100, 1000, 10000, 100000] } ]**
    - **RR=Ridge()** \*Ridge Regression object
    - **Grid1=GridSearchCV(RR, parameters1, cv=4)** \*Grid search CV object.
    - **Grid1.fit(x\_data[[‘horsepower’, ‘curbe-weight’, ‘engine-size’, ‘highway-mpg’]], y\_data)** \*Train the model
    - **Grid1.best\_estimator\_ \***Find the best values for the free parameters,
    - **Scores = Grid1.cv\_results\_**
    - **Scores[‘mean\_test\_score’]**
  + **Grid search normilization**
    - Ex: **parameters = [{‘alpha’ : [1, 10, 100, 1000], ‘normalize’ : [True, False]}]**

**Table

Description automatically generated**

* **Code:**
  + **From sklearn.linear\_model import Ridge**
  + **From sklearn.model\_selection import GridSearchCV**
  + **Parameters2 = [ { ‘alpha’ : [0.001,0.1,1,10,100], ‘normalize’ : [True, False] } ]**
  + **RR = Ridge()**
  + **Grid1 = GridSearchCV(RR, parameters2, cv=4)**
  + **Grid1.fit(x\_data[[‘horsepower’, ‘curb-weight’, ‘engine-size’, ‘highway-mpg’]], y\_data)**
  + **Grid1.best\_estimator\_**
  + **Scores = Grid1.cv\_results**
* **To print**
  + **For param,mean\_val, mean\_test inzip(scores[‘params’], scores[‘mean\_test\_score’], scores[‘mean\_train\_score’]):**
    - **Graphical user interface, text, application, email

      Description automatically generatedPrint(param, “R^2 on test data:”, mean\_val, “R^2 on train data:” , mean\_test)**