Data Analysis with Python

Módulo 1 - Introducción a Data Science

[ ] ↳ *21 celdas ocultas*

Módulo 2 - Data Wrangling

[ ] ↳ *24 celdas ocultas*

Módulo 3 - Exploratory Data Analysis

[ ] ↳ *24 celdas ocultas*

Módulo 4 - Model Development

↳ *9 celdas ocultas*

Módulo 5 - Model Evaluation and Refinement

Objectives

- Model evaluation
- Over-fitting, under fitting and model selection
- Ridge regression
- Grid search
- Question:
    - How can you be certain your model work for the real world and performs optimally?

Model evaluation

Model Evaluation tells us how our model preforms in the real world.

In-sample evaluation tell us how well our model will fit the data used to train it

Problem?

- It does not tell us how well the trained model can be used to predict new data

Solution?

- In sample data or training data
- Out of sample data evaluation or test set

Data:

- Split dataset into:
    - Training set (70%)
    - Testing set (30%)
- Build and train the model with a training set
- Use testing set to asses the performance of a predictive model
- When data have completed testing our model we should use all the data to train the model to get the best performance

Function train_test_split()

```
from sklearn.model_selection import traint_test_split

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, tes_size, random_state=0)
```

- x_data: features or indepent variables
- y_data: dataset target: df['price']
- x_train, y_train: parts of the availabledata as training set
- test_size: percentage of the data for testing (here 30%)
- random_state: number generator used for random sampling

Generalization performance

- Generalization error is measure of how well our data does at predicting previously unseen data
- The error we obtain using our testing data is an approximation of this error

Cross validation

- The output is the R^2 average on the test data for each of the selected folds (cv)
- Most common out of sample evaluation metrics

- More effective use of data (each observation is used for both training and testing)

Function cross_val_score()

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(lr, x_data, y_data, cv=3)

np.mean(scores)
```

Function cross_val_predict()

```
from sklearn.model_selection import cross_val_predict

yhat = cross_val_predict(lr2e, x_data, y_data, cv=3)
```

- It returns the prediction that was obtain for each element when it was in the test set
- Has a similar interface to cross_val_score()

Overfitting, Underfitting and Model Selection

- Underfitting: where the model is too simple to fit the data
- Overfitting: where the model is too flexible and fits the noise rather than the function

We can calculate different R-squared values as follows:

```
Rsqu_test = [] #1
order = [1, 2, 3, 4] #2

for n in order: #3
  pr = PolynomialFeatures(degree=n) #4
  x_train_pr = pr.fit_transform(x_train[['horsepower']]) #5
  x_test_pr = pr.fit_transform(x_test[['horsepower']]) #5
  lr.fit(x_train_pr, y_train) #6
  Rsqu_test.append(lr.score(x_test_pr, y_test)) #7
```

1. First, we create an empty list to store the values

2. We create a list containing different polynomial orders
3. We then iterate through the list using a loop
4. We create a polynomial feature object with the order of the polynomial as a parameter
5. We transform the training and test data into a polynomial using the fit transform method
6. We fit the regression model using the transformed data
7. We then calculate the R-squared using the test data and store it in the array

Ridge Regression

```
from sklearn.linear_model import Ridge #1
RigeModel = Ridge(alpha=0.1) #2
RigeModel.fit(X, y) #3
Yhat = RigeModel.predict(X) #4
```

Ridge regression prevents over-fitting. we will focus on polynomial regression for visualization, but over-fitting is also a big problem when you have multiple independent variables or features.

Ridge regression controls the magnitude of these polynomial coefficients by introducing the parameter alpha.

Alpha is a parameter we select before fitting or training the model. Each row in the following table represents an increasing value of alpha.

Alpha must be selected carefully. If alpha is too large, the coefficients will approach zero and under-fit the data. If alpha is zero, the over-fitting is evident.

Steps to follow:

1. Import ridge from sklearn linear models
2. Create a Ridge object using the constructor
    - The parameter alpha is one of the arguments of the constructor
3. We train the model using the fit method
4. To make a prediction, we use the predict method

In order to determine the parameter alpha, we use some data for training. We use a second set called validation data; this is similar to test data, but it is used to select parameters like alpha. We start with a small value of alpha, we train the model, make a prediction using the validation data, then calculate the R squared and store the values. Repeat the value for a larger value of alpha.

We select the value of alpha that maximizes the R squared

- *Note that we can use other metrics to select the value of alpha like mean squared error.*

Grid Search

Grid search allows us to scan through multiple free parameters with few lines of code.

Grid search takes the model or objects you would like to train and different values of the hyperparameters.

It then calculates the mean square error or R squared for various hyperparameter values, allowing you to choose the best values.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters1 = [{'alpha':[0.001, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000]}]

RR = Ridge()
Grid_1 = GridSerachCV(RR, parameters1, cv=4)

Grid_1.best_estimator_

Scores = Grdi.cv_results_
Scores['mean_test_score']
```

1. First, we import the libraries we need including Grid Search CV, the dictionary of parameter values
2. We create a ridge regression object or model
3. We then create a Grid Search CV object; the inputs are the ridge regression object, the parameter values and the number of folds
    - We will use R squared; this is the default scoring method
4. We fit the object
5. We can find the best values for the free parameters using the attribute best estimator
6. We can also get information like the mean score on the validation data using the attribute cv result

One of the advantages of Grid search is how quickly we can test multiple parameters.

For example, Ridge regression has the option to normalize the data. To see how to standardize, see Module 4.

```
parameters = [{'alpha': [1, 10, 100, 1000], 'normalize': [True, False]}]
```

The term alpha is the first element in the dictionary, the second element is the normalize option.

The key is the name of the parameter. The value is the different options, in this case, because we can either normalize the data or not, the values are true or false, respectively.

The Dictionary is a table or grid that contains two different values.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV

parameters_2 = [{'alpha': [0.001, 0.01, 0.1, 1, 10, 100], 'normalize': [True, False]}]

RR = Ridge()
Grid1 = GridSerachCV(RR, parameters1, cv=4)
Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_data)

Grid1.best_estimator_
scores = Grid1.cv_results_
```

We can print out the score for the different free parameter values.

```
for param, test_mean_val, mean_test inzip(scores['params'], socres['mean_test_score'], scores['mean_t
  print(param, "R^2 on test data: ", mean_val, "R^2 on train data: ", mean_test)
```