Module 05: Model Evaluation

Learning Objectives:

→ Model Evaluation
→ Over-fitting, Under-fitting and Model Selection
→ Ridge Regression
→ Grid Search
→ Question:
  * How can you be certain your model works in real world and performs optimally.

Model Evaluation

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

- Problem?
  - It does not tell how will the trained model can be used to predict new data

- Solution? - In-sample data or training data,
  - Out-of-sample evaluation or test set

~~Model~~ Training / Testing Sets

Data

- Split dataset into:
  - → Training set (70%)
  - → Testing set (30%)

- Build and train the model with a training set
- Use testing set to assess the performance of a predictive model
- When we have completed testing our model we should use all the data to train the model to get the best performance.

Function train_test_split()

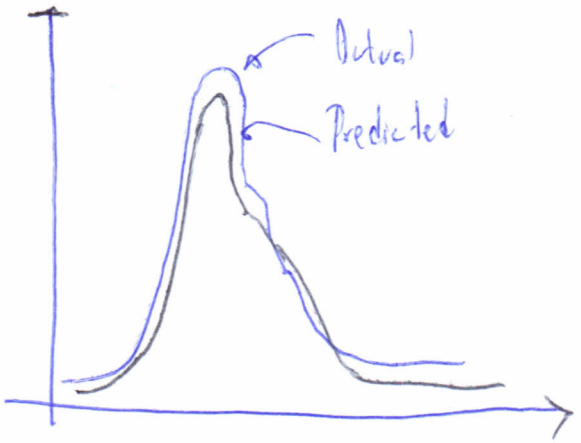
- Split data into random train and tests subsets

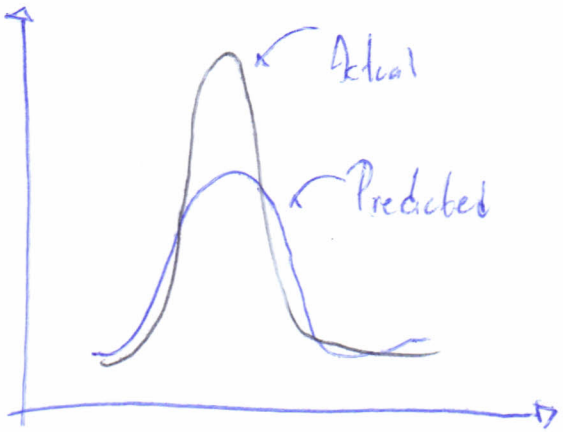
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.3,
                                                    random_state=0)
```

- x_data ⟹ features or independent variable
- y_data ⟹ dataset target : df['price']

Generalization Performance

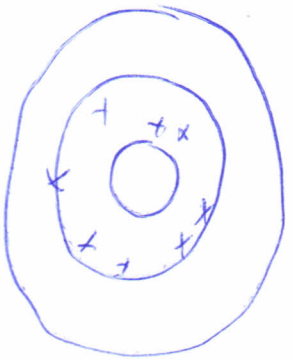
- Generalization error is measure of how well our data does at predicting previously unseen data
- The error ~~eb~~ we obtained using our testing data is an approximation of this error.



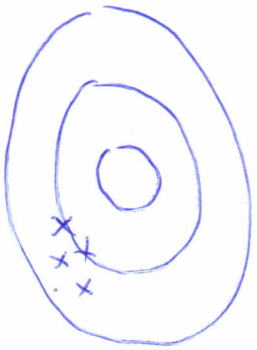


Lots of Training Data

Model



True Generalization Error

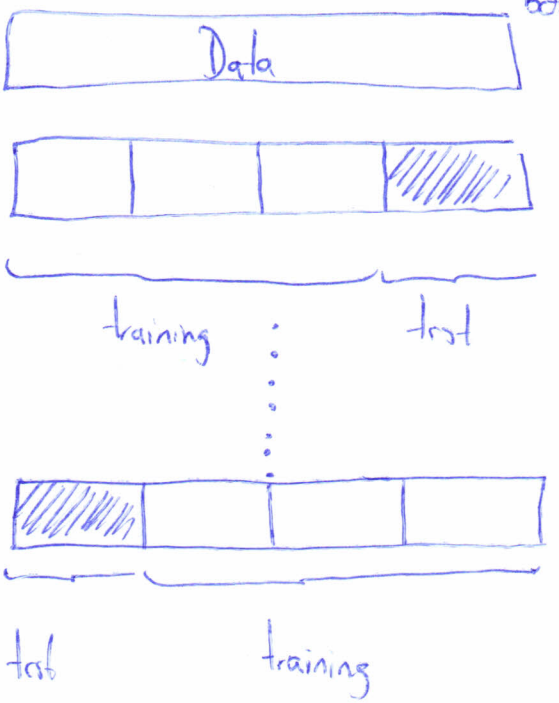


Good Precision

Cross Validation to this problems

Cross-Validation

- Most common out-of-sample evaluation metrics
- More effective use of data (each observations is used for both training and testing)



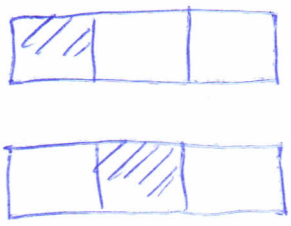
Function cross_val_score()

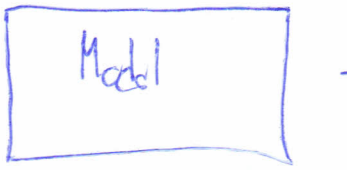
```
from sklearn.model_selection import cross_val_score

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

predictor — variable data — target variable — number of partitions

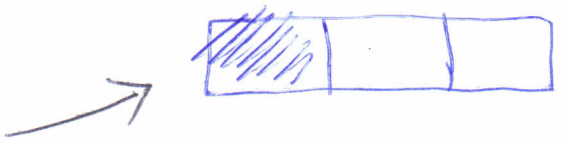
```
np.mean(scores)
```

Function cross_val_score()





Model $\Longrightarrow R^2 = \left(1 - \frac{\text{MSE of regression line}}{\text{MSE of } \bar{y}}\right)$



Array

Scores

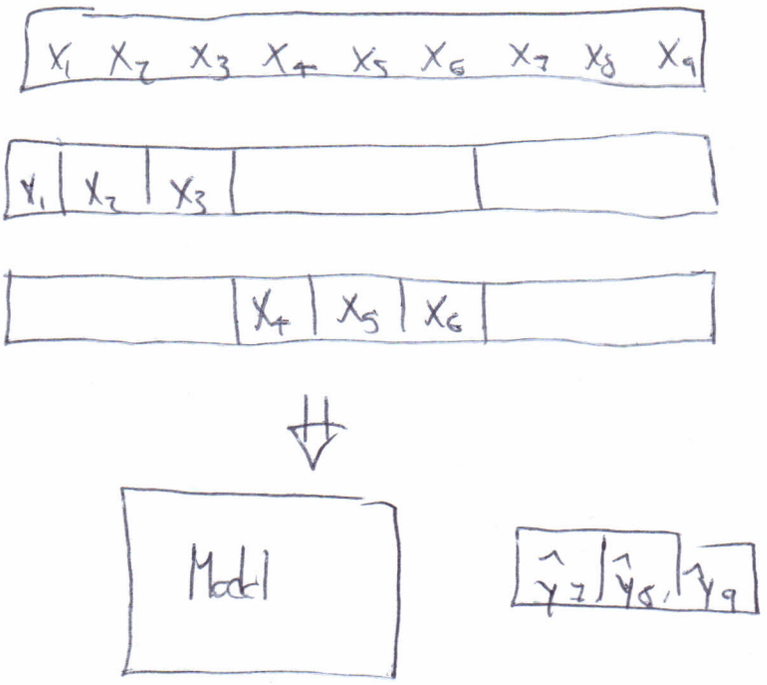
Function cross_val_predict()

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

```
from sklearn.model_selection import cross_val_predict

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

The Output is a prediction

Function cross_val_predict()

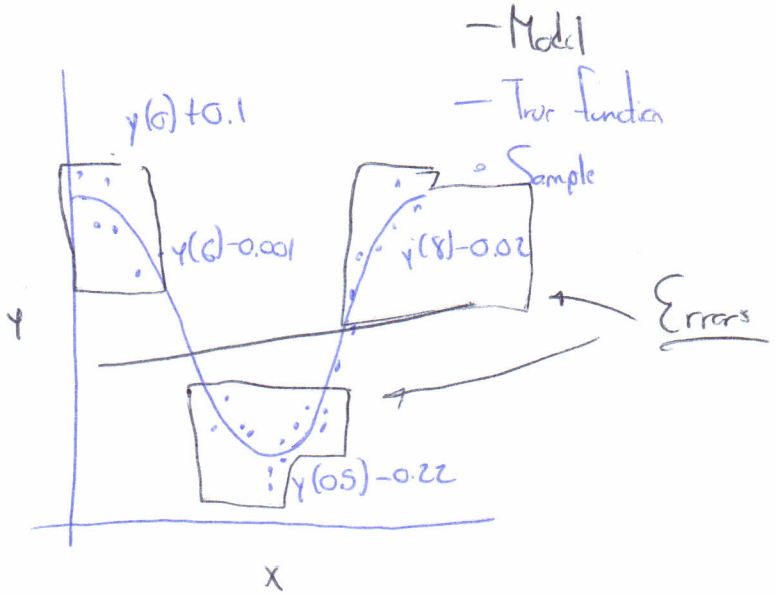


Predictions are stored in array

Overfitting, Underfitting and Model Selection

Model Selection

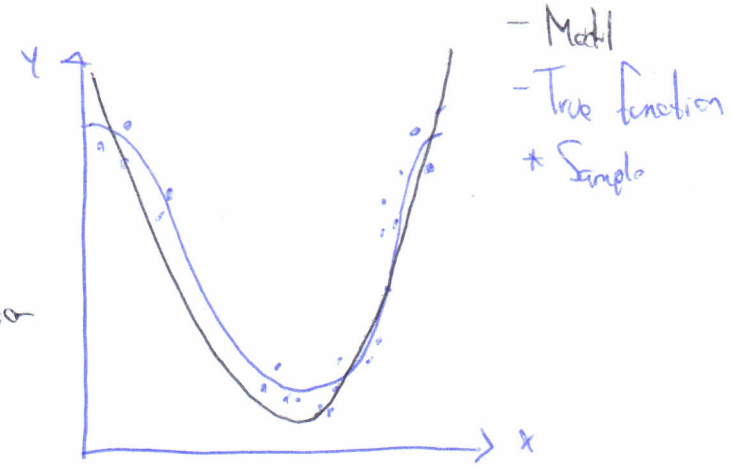
$y(x) + \text{noise}$



$y = b_0 + b_1 x$

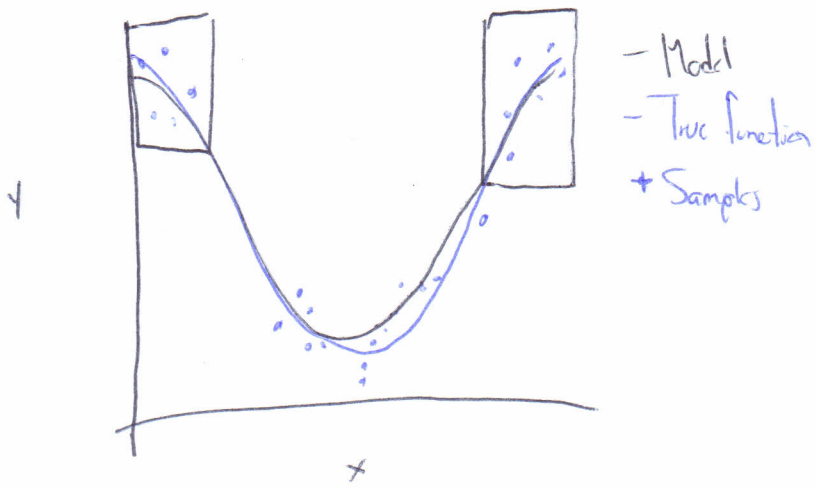
Underfitting $\Rightarrow$ The model is too simple to fit the data.

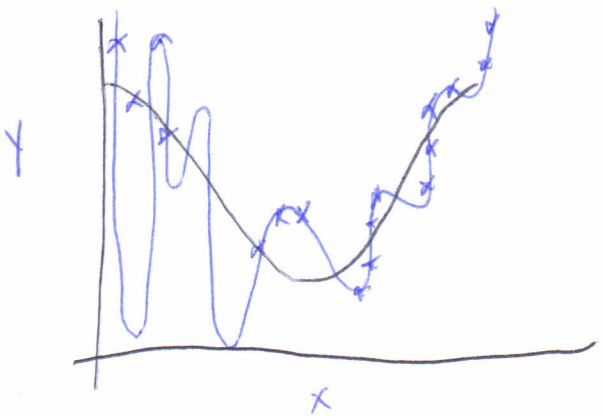
$y = b_0 + b_1 x + b_2 x_2$



Increase the order $\Rightarrow$ fits better but the model is still not flexible enough and exhibits underfitting

$$\hat{y} = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + b_4 x^4 + b_5 x^5 + b_6 x^6 + b_7 x^7 + b_8 x^8$$





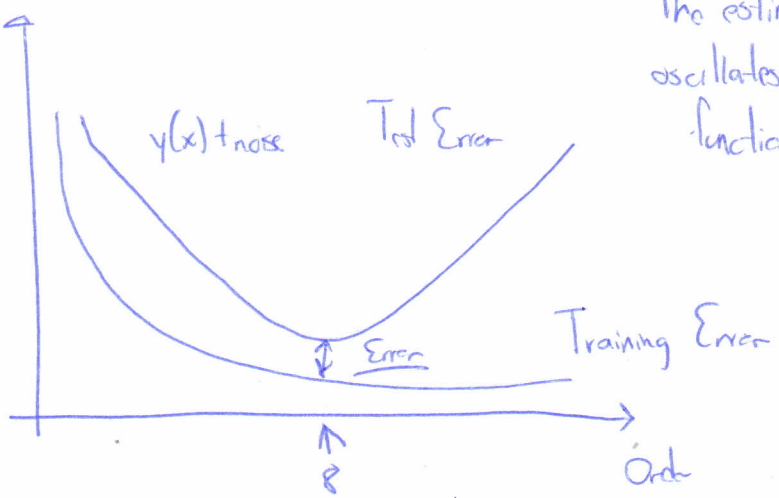
The estimated model is too flexible and fits the noise rather that the function.

Over-fitting

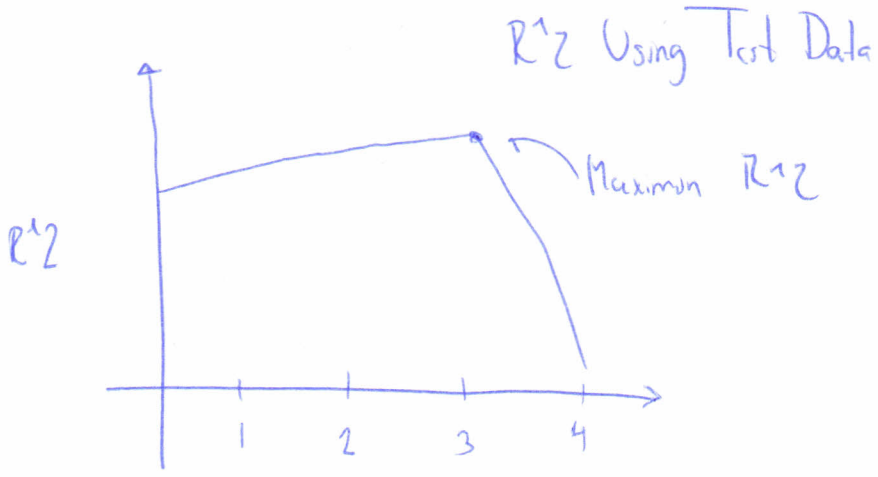
⇓

The estimated function oscillates not tracking the function.

Model Selection



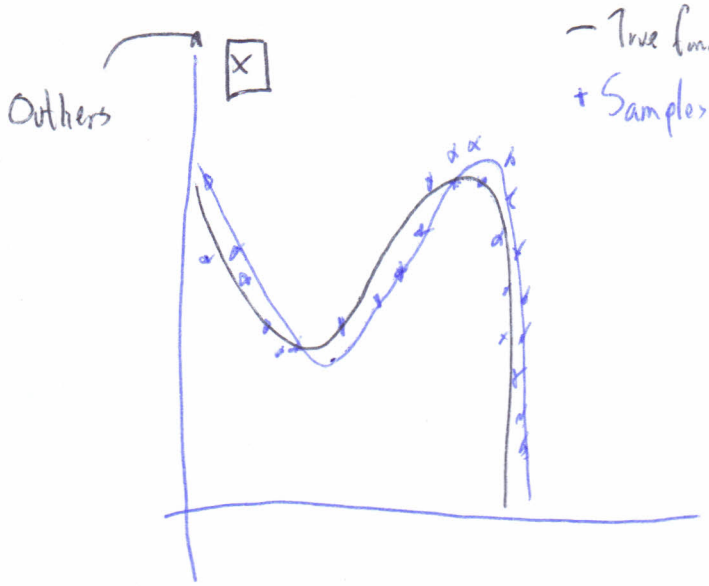
Model Selection




```
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(y_test[['horsepower']])

lr.fit(x_train_pr, y_train)

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

Ridge Regression

$$y = 1 + 2x - 3x^2 - 4x^3 + x^T$$



$$\hat{y} = 1 + 2x - 3x^2 - 2x^3 - 12x^4 - 40x^5 + 80x^6 + 71x^7 - 141x^8 - 38x^9 + 75x^{10}$$

Coefficients with a very large magnitude

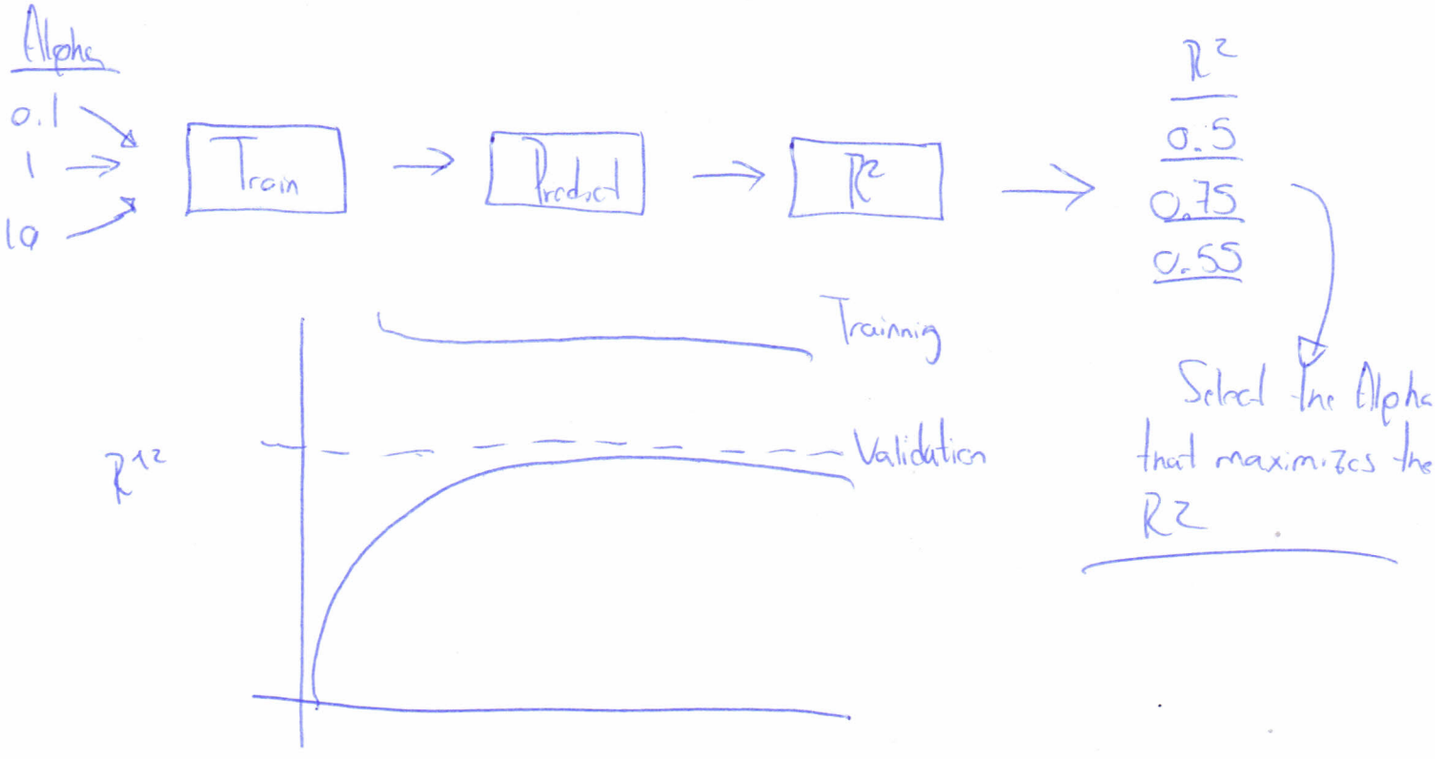
Ridge Regression controls the magnitude of these polynomial coefficients.

Alpha
0
0.001
0.001
1
10

Parameter we select before fitting or training the model.

Ridge Regression

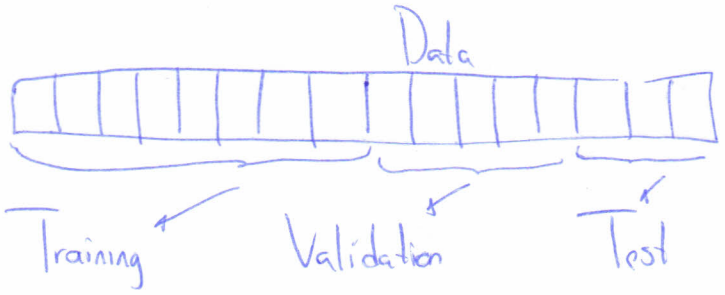
```
from sklearn.linear_model import Ridge

RigeModel = Ridge(alpha=0.1)

RigeModel.fit(x, y)

Yhat = RigeModel.predict(x)
```



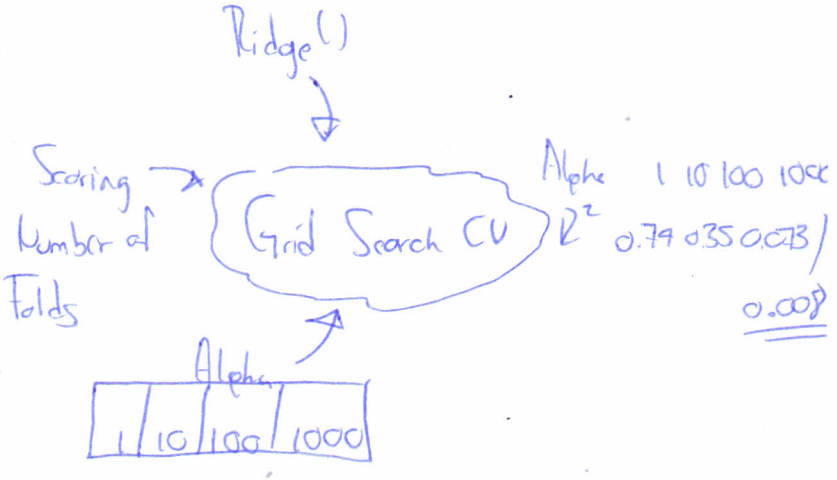
Overfitting ⇒ Worse if we have a lots of features

Grid Search

- In the last section, the term alpha is Ridge Regression is called a hyperparameter
- Scikit-learn has a mean of automatically iterating over these hyperparameters using cross-validation called Grid-Search
- Select the hyperparameters that minimize the error.



parameters = [{'alpha': [1, 10, 100, 1000]}]



Grid Search

```
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()

Grid1 = GridSearchCV(RR, parameters1, cv=4)

Grid1.fit(x_data[['horsepower', 'curb-weight', 'engine-size', 'highway-mpg']], y_data)

Grid1.best_estimator_

scores = Grid1.cv_results_
scores['mean_test_scores']
```

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

Alpha	1	10	100	1000
Normalize	True	True	True	True
	False	False	False	False

```
Ridge()
```

Ridge ()

⟹

Alpha	1	10	100	1000
True	0.69	0.32	0.17	0.17
False	0.67	0.66	0.66	0.64

```
for param, mean_val, mean_test in zip(scores['params'], scores['mean_test_score'], scores
    ['mean_train_scores'])

print(param, "R^2 on test data:", mean_val, "R^2 on train data:", mean_test)
```