Model evaluation

In-sample evaluation tells us how well our model fits the data already given to train it. It does not give us an estimate of how well the trained model can predict new data. The solution is to split our data up, use the In-sample data or training data to train the model.

Training and testing

```
y_data = df['price']

x_data=df.drop('price',axis=1)

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])

x_train1, x_test1, y_train1, y_test1 = train_test_split(x_data, y_data, test_size=0.4, random_state=0)
```

Cross Validation

```
from sklearn.model_selection import cross_val_score

Rcross = cross_val_score(lre, x_data[['horsepower']], y_data, cv=4)

print("The mean of the folds are", Rcross.mean(), "and the standard deviation is" , Rcross.std())

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

Ridge regression

Ridge regression prevents over-fitting.

```
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']])
```

```
from sklearn.linear_model import Ridge

RigeModel=Ridge(alpha=1)
RigeModel.fit(x_train_pr, y_train)

yhat = RigeModel.predict(x_test_pr)

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

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)

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')
plt.legend()
```

Grid search

Grid search allows us to scan through multiple free parameters with few lines of code. Parameters like the alpha term discussed in the previous video are not part of the fitting or training process. These values are called hyperparameters.

Scikit-learn has a means of automatically iterating over these hyperparameters using cross-validation. This method is called Grid search. 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.model_selection import GridSearchCV
parameters1= [{'alpha': [0.001,0.1,1, 10, 100, 1000, 10000, 100000, 100000]}]
parameters1

RR=Ridge()
RR

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

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