## Objectives

Estimated Learning Time: 3h 35m

There are a few best practices to avoid overfitting of your regression models. One of these best practices is splitting your data into training and test sets. Another alternative is to use cross validation. And a third alternative is to introduce polynomial features. This module walks you through the theoretical framework and a few hands-on examples of these best practices.

Learning Objectives:

* Realize the importance of having a test set to avoid overfitting
* Practice using the train\_split function to split your data into training and testing sets
* Identify common cross validation approaches
* Recognize the tradeoff between model complexity and prediction error
* Assess whether introducing polynomial features improves the error metrics of your linear regression

Learning goals

* Splitting data into training and testing samples
* Cross-validation approaches
* Model complexity vs. Errror

Training data – fit the model

Test data – measure performance

* Predict label with model
* Compare the actual value
* Measure error

Training data --> x\_train y\_train --> model(X\_train, y\_train).fit() --> model

Test data --> x\_test --> model .predict(X\_test) --> y\_predict

\\_\_ Y\_test(actuals) \_\_> error\_metric(Y\_test, y\_predict) --> test error

Train-test split: the syntax

Import the train and test split function

From sklearn.mode\_selection import train\_test\_split

Split the data and put 30% into the test set

Train, test = train\_test\_split(data, test\_size=0.3)

Othe method for splitting data

From sklearn.mode\_selection import shufflesplit

Data.dtypes.value\_counts() - will give you unique data types value

Unordered categories should be one-hot-encoded

Set mask to all objects in dataset

**What is lambda**

Onehot encoder is same but better than pd.get\_dummies

**Prediction and interpretability**

Multicollinearity

Question 3 is focusing on predictability

From sklearn.preprocessing import OneHotEncoder

Num\_ohc\_cols.index

Col = ‘Neighborhood’

Data\_ohc=data.copy()

Ohc = OneHotEncoder()

New\_data = ohc.fit\_transform(data\_ohc[[col]])

The two brackets above will make it a data frame (2 dimensional, instead of one deminsional)

* Data[‘neighborhood’].shape - will output one column
* Data[[‘Neighborhood’]].shape - will output 2 columns

**Sparse matrix** – used to save memory

We drop the original column from ds because we have the one hot encoded data of that column

Pd.DataFrame converts an array into a dataframe

Take copy of data and concat with our new dataframe

* axis=1 means we are putting it to the right

Then look at difference between the new dataframe from the original dataframe

This will be 215 and equal our 215 from earlier

Then we need to drop all the original columns that were strings.

**Question 4.**

2 different sets:

* 37 diff features
* 294 features

Train\_test\_split

Y is set to our target column

Random state is to get the same split everytime

X and y train will be paired together to fit our model

965 rows or 70% of rows = x\_train\_ohc.shape

Check indicies to make sure they split is the same

Linear regression for how well we do on train and test sets

Fit model on original x train and y train without onehot encodding

Lr predict to see how we will do on the x\_train

Lr predict on x test to see how well

Series is like a column

Lower error for onehot encoded for train set

Test set performed worse with one hot encodding

The reason for this – is most likely from overfitting

Big gap between train set error and test set error is an indicator of overfitting

**Question 5.**

Fit transform only on the training set

Then transform on the test set

Scaler will point to key standard

Lr is fit to training set

We get predictings on test sets by calling r on test

We get same error for each which is the same for the previous test set

Scaling wont normally effect error on linear regression

Alpha=/05 makes it slightly transparent

The closer the predictions are to the line the better our predictions are.

**Quiz:**

Another common term for the testing split is:

* Validation split

Complete the following sentence: The training data is used to fit the model, while the test data is used to:

* Measure error and performance of the model

Select the option that has the syntax to obtain the data splits you will need to train a model having a test split that is a third the size of your available data.

* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33)

**Cross Validation**

Beyond a single test set: cross validation

With cross validation we split into multiple pairs of test train sets

Average cross validation results

Error metrics relationship to complexity

With train set - The larger the complexity the lower the training error

With cross validatoin – the graph of complexity to error is a u shape

Left side of curve - Underfitting training a cross validation error are both high

Right side of graph is overfitting – training error is low, cross validation is high

Just right point is middle of graph where training and cross validation errors are both low

Cross validation approaches

K-fold validation: using each of k subsamples as a test sample

Leave one out cross validation: using each obseration as a test sample

Stratified cross validation: k-fold cross validation with representative samples

Cross validation: the syntax

Import the cross validation-function

from

Sklearn.model\_selection import cross\_val\_score

Perform cross-validation with a given model

Cross\_val = cross\_val\_scorel(model, X\_data, y\_data, cv=4, scoring=’neg\_mean\_squared\_error’)

The cv is the number of splits

Other methods for cross validation:

From sklearn.model\_selection import kfold, stratifiedkfold

**Cross validation demo**

Preprocessing with pipeline

Kflods to split data

Cross validation

Pt.2

Preprocessing step os standardSclaer()

For loop to get the test and train sets for x and y

Then scale x train

Then fit linear regression on our training set

Transform test set save to x\_test\_s

Apeend on new scores to our scores list.

Piprline chain on multiple operators as long as they have a fit method. So the output of one can be the input of the next

They need fit transform and the lastone has to be fit

Oinstead of the for loop to get prodictions we will use cross val predict

Fkf

For my estimator, my intitial value of x and initial y values, I want to pass in the folds from kf

Run length of predictions should be same length of df

Cross val predict didn’t actually fit the model at any step. It gives all the outputs but with 3 diff models.

We learn our parameters but have to choose our hyper parameters

Hyperparameters are used to optimize the model performance

Geomspace every value between1 and 1000 will be a multiple of the previous value

The higher the alpha is the less complexx your model is

We want to see which alpha scores lead to the highest score

Scalke data before you use ridge or lasso regression

We do polynomial features first then

Ridge and lasso reduce the complexity of

Goal is to find the optimal hyperparameters

Interpretability of lr all features are on the same scale

Because we are worried about interpretability and not predictability we bring in all our training data

.name steps alowws us to acces a dictionary

Grid seach cv

Do what we did above a lot easier with gridsearch cv

We will look at polynomial features and alpha values

**Polynomial Regression**

Goals:

* extending linear regression
* Using poolynomial features to capture nonlinear effects
* Other models that can be used for regression and classification

Capture higher order features of data by adding polynomial features

‘linear regression’ means linear combinations of features

Addition of polynomial features

Can also include variable interactions:

How is the correct functional form chose:

* Check relationship of each variable or with outcome.

Enhancing the linear model

Adjusting the standard linear approach to regression by adding polynomial features is one of many approaches to dealing with fundamental problems:

* Prediction
* Interpretation

As we move into model evaluation, keep in mind that the same tools are useful for evaluating a wide variety of regression and classification problems.

Extending the linear model

In addition to polynomial features, we will also examine several additional variants of standard models, using many of both regression and classification. Some examples include:

* Logistic regression
* k-nearest neighbors
* Decision trees
* Support vector machines
* Random forests
* Ensemble methods
* Deep learning approaches

Polynomial features: the syntax

Import the class containg the transfomration method

From sklearn.preprocessing import PolynomialFeatures

Create an instance of the class

PolyFeat = polynomialFeatures(degree=2)

Create the polynomial features and then transform the data

PolyFeat = polyfeat.fit(X\_data)

X\_poly = polyFeat.transform(X\_data)

## End of module review: Data Splits and Cross Validation

### **Training and Test Splits**

Splitting your data into a training and a test set can help you choose a model that has better chances at generalizing and is not overfitted.

The training data is used to fit the model, while the test data is used to measure error and performance.   
Training error tends to decrease with a more complex model. Cross validation error generally has a u-shape. It decreases with more complex models, up to a point in which it starts to increase again.

### 

### **Cross Validation**

The three most common cross validation approaches are:

* k-fold cross validation
* leave one out cross validation
* stratified cross validation

### 

### **Polynomial Regression**

Polynomial terms help you capture nonlinear effects of your features.

Other algorithms that help you extend your linear models are:

* Logistic Regression
* K-Nearest Neighbors
* Decision Trees
* Support Vector Machines
* Random Forests
* Ensemble Methods
* Deep Learning Approaches