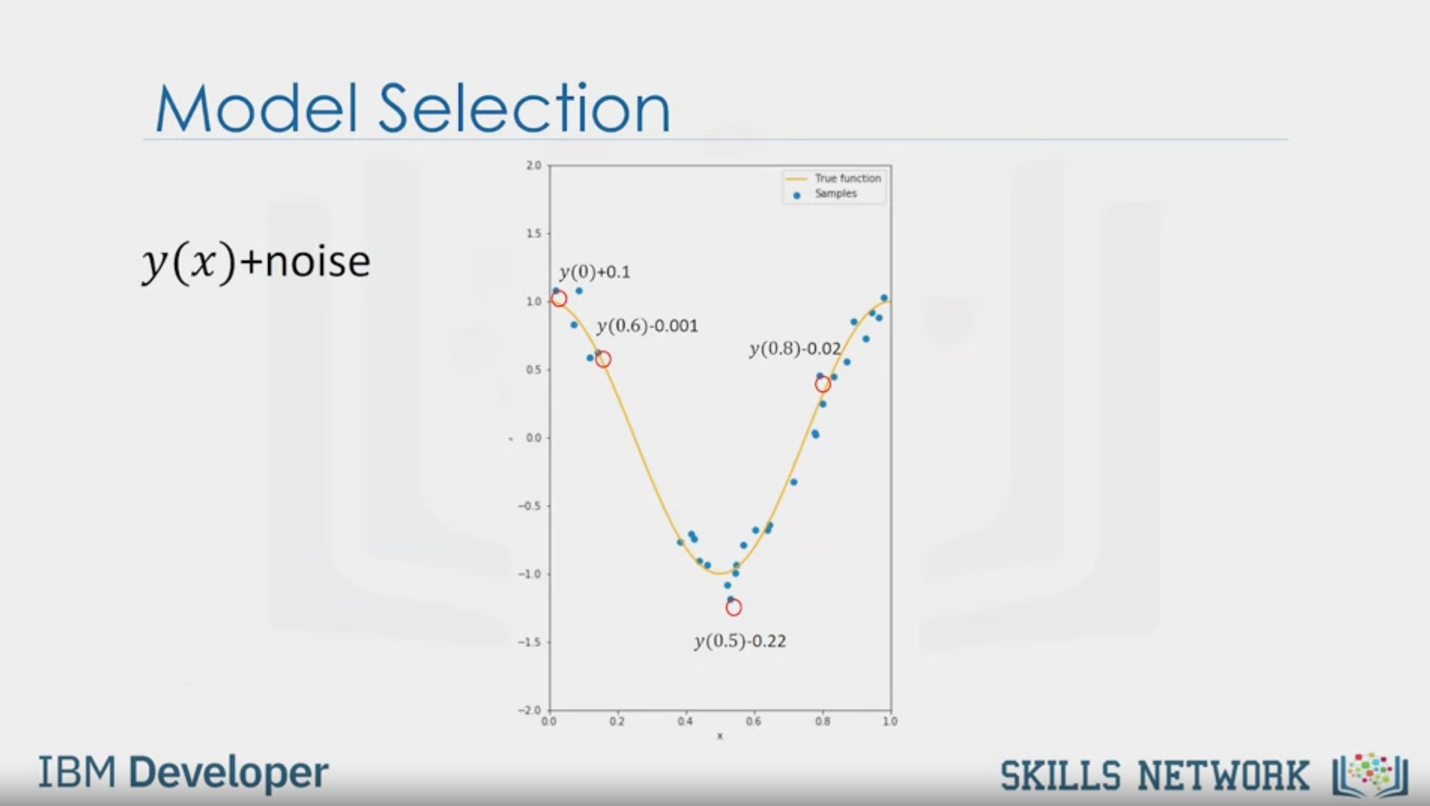


Overfitting, Underfitting and

Model Selection

IBM Developer

SKILLS NETWORK



Model Selection

y(x)+noise

y (0)+0.1

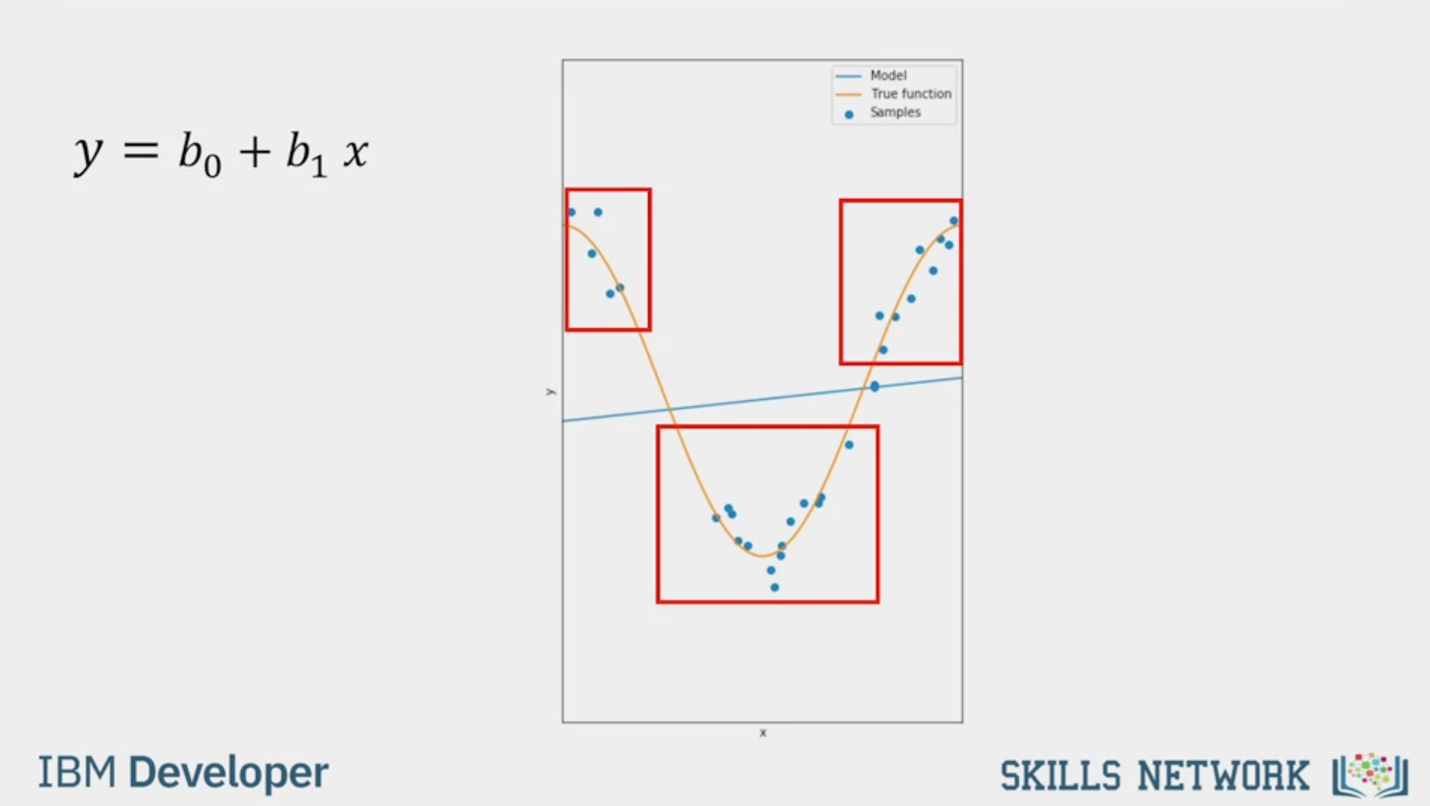
y(0.6)-0.001

v(0.5)-0.22

y(0.8)-0.02

IBM Developer

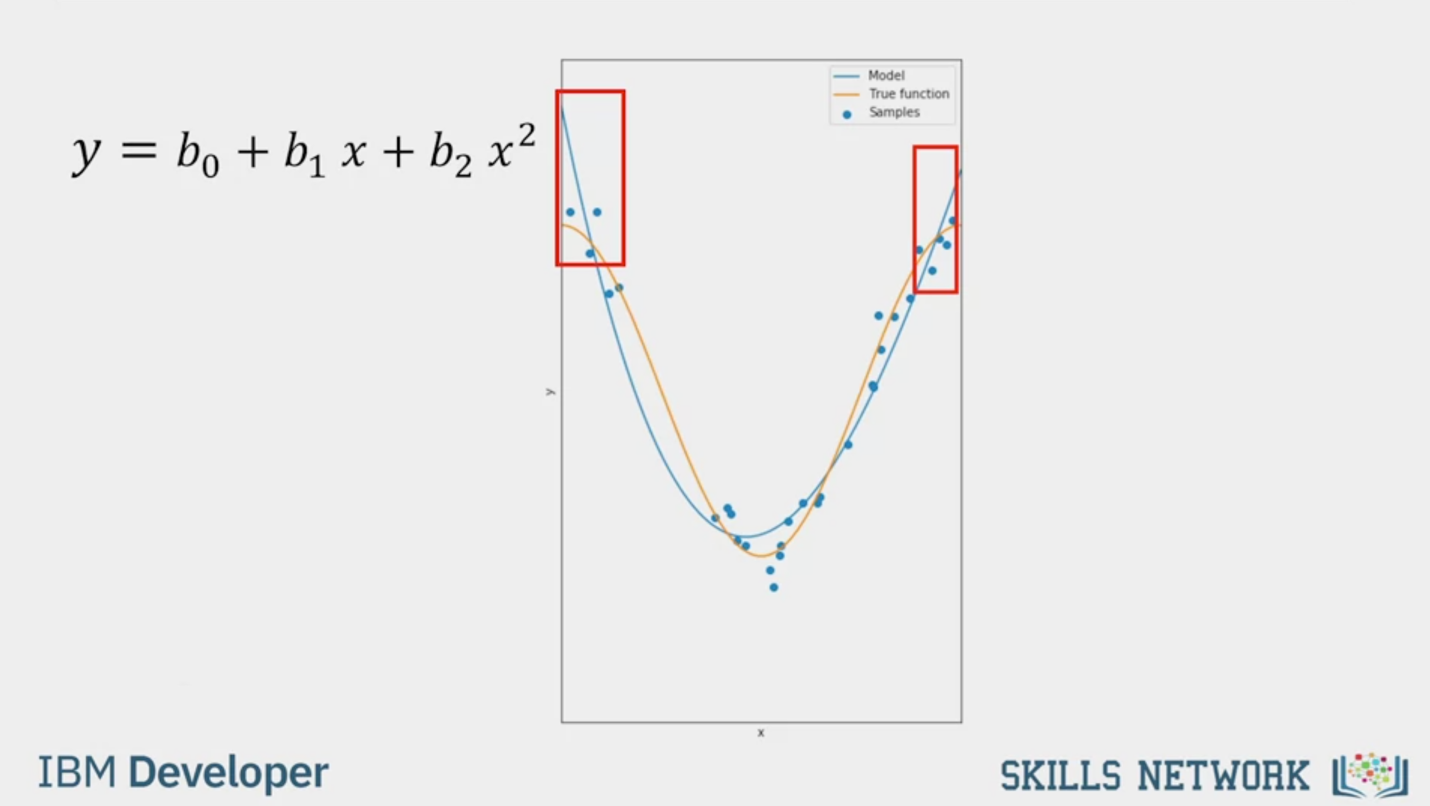
SKILLS NETWORK



y = b0 + b1x

IBM Developer

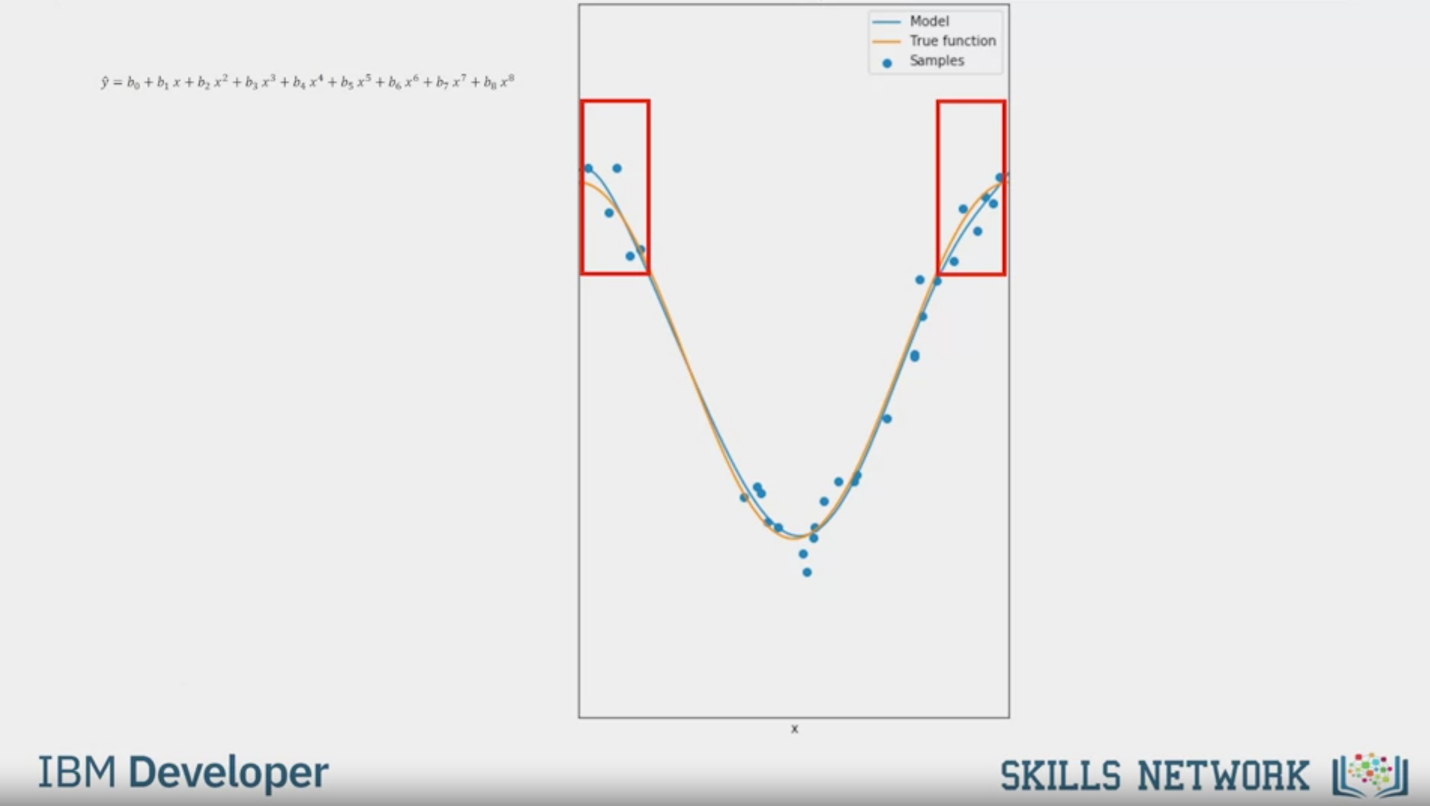
SKILLS NETWORK



y = b0 + b1x + b2x2

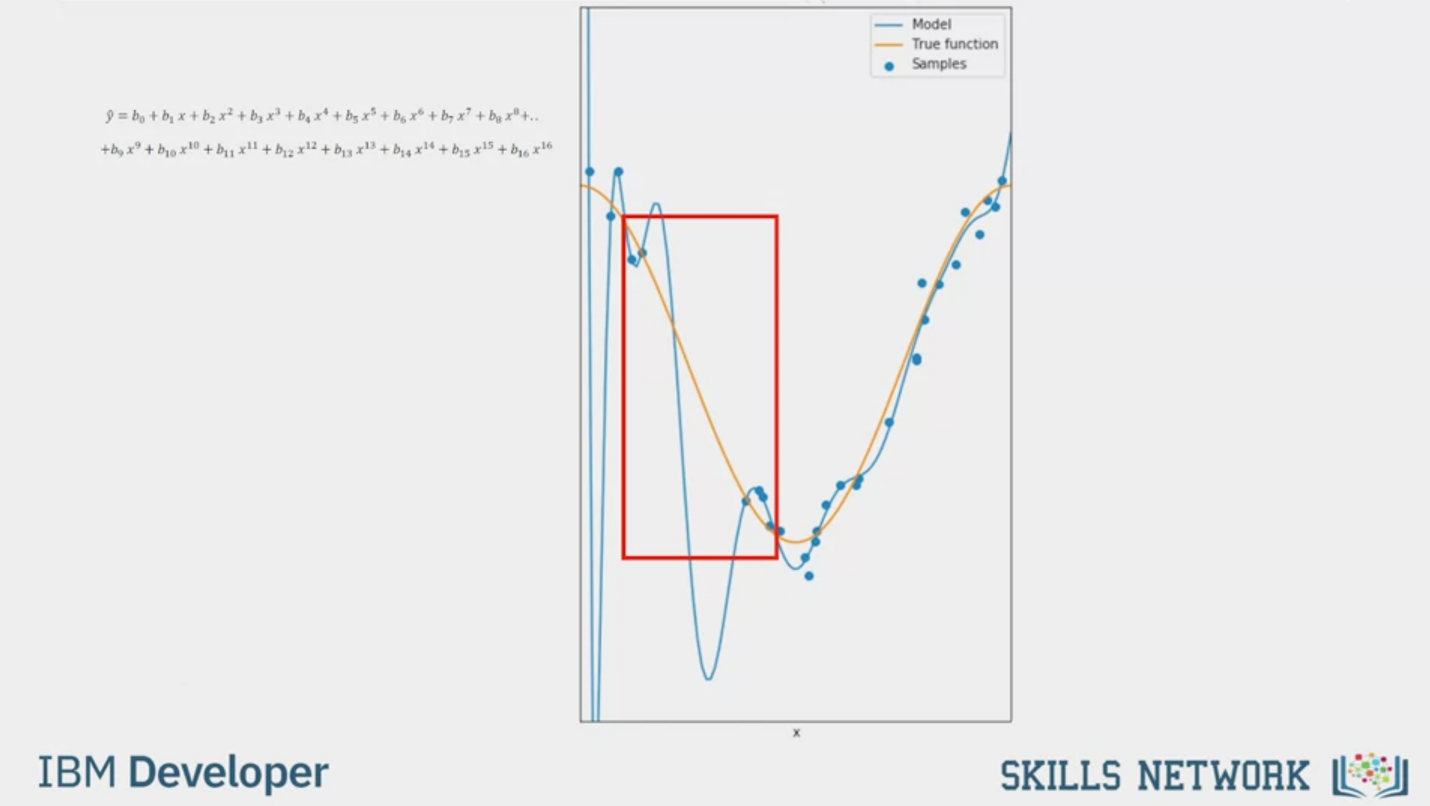
IBM Developer

SKILLS NETWORK



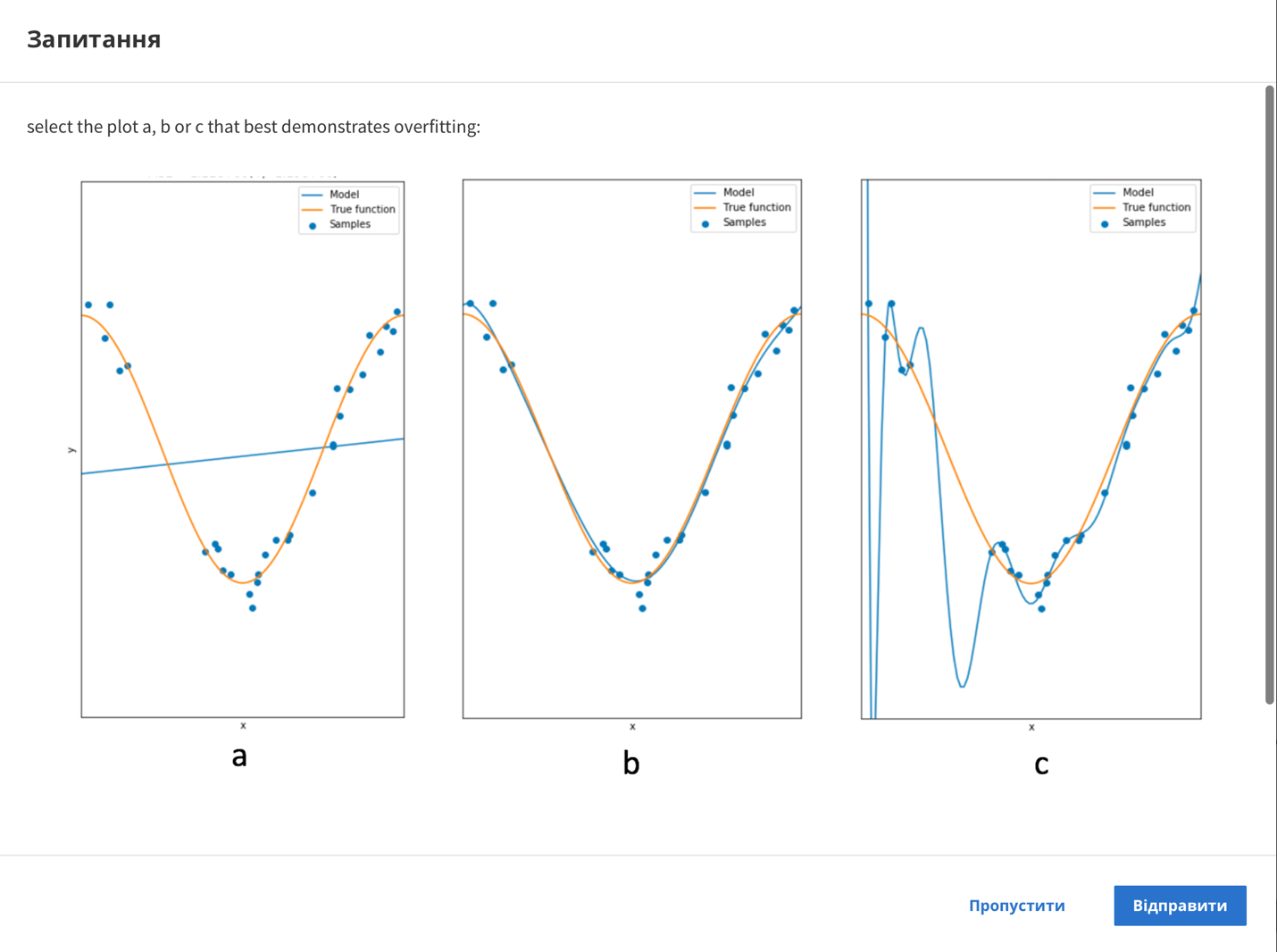
IBM Developer

SKILLS NETWORK



IBM Developer

SKILLS NETWORK



Question

select the plot a, b or c that best demonstrates overfitting:

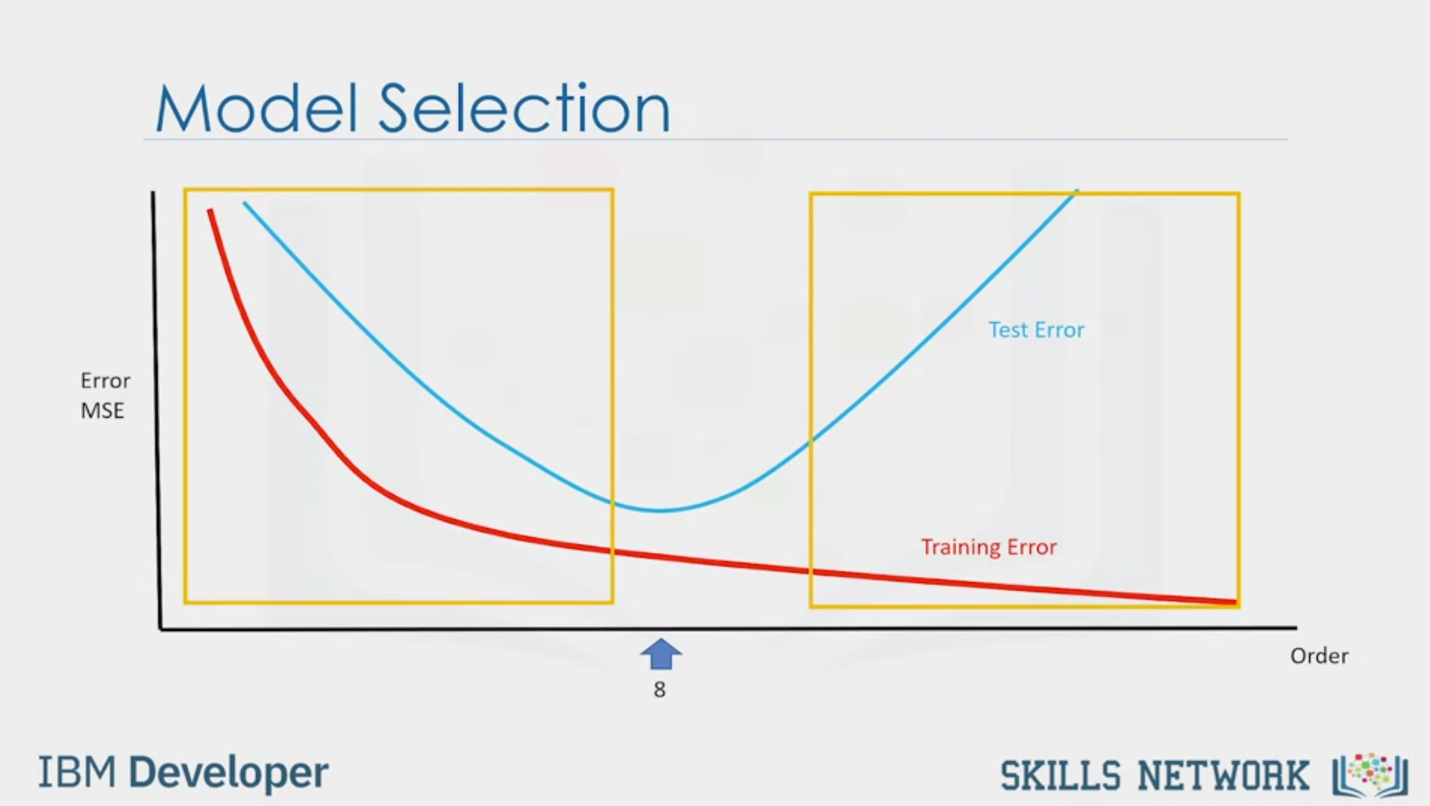
a

b

c

Skip

Send



Model Selection

Error

MSE

8

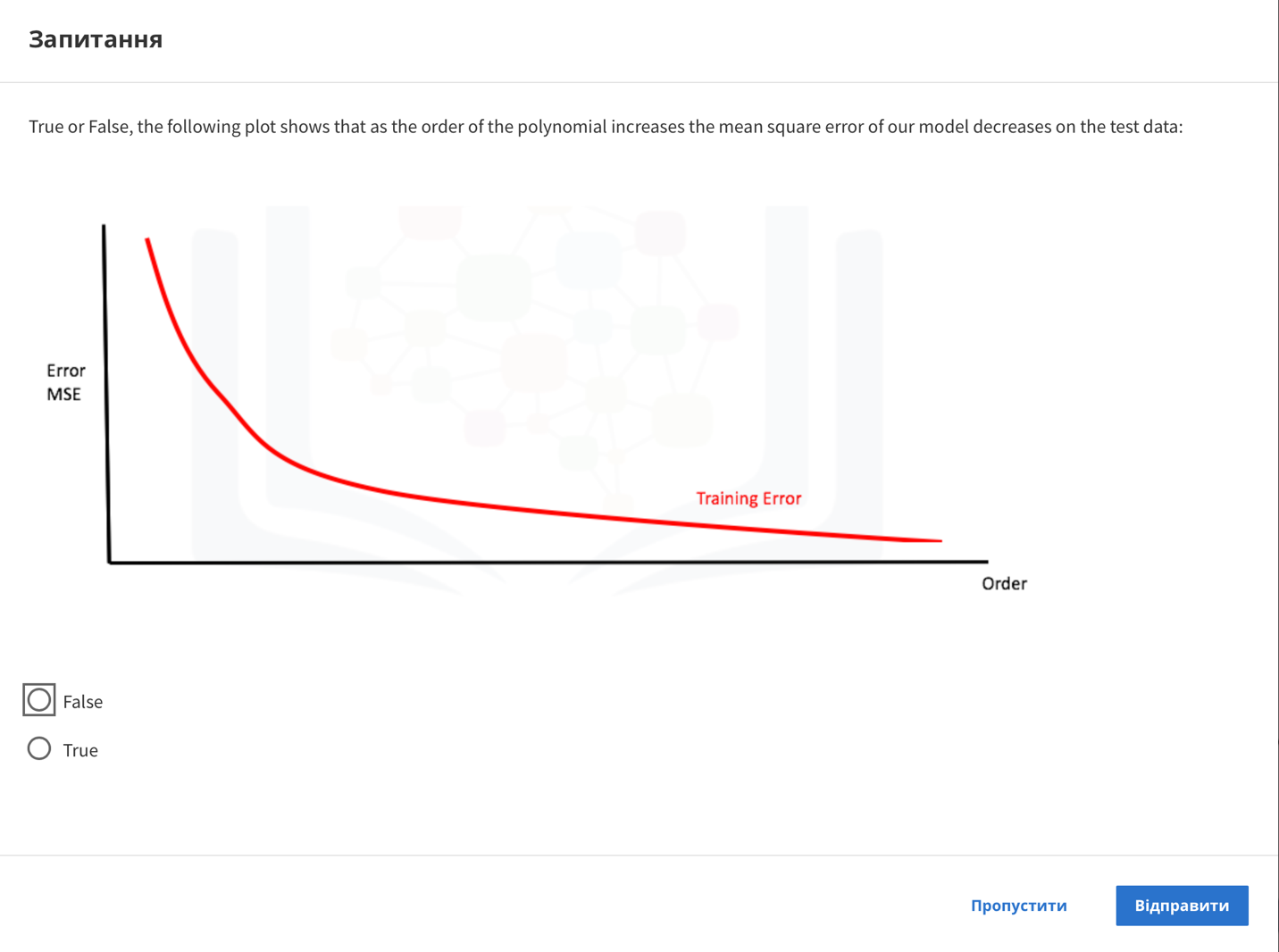
Test Error

Training Error

Order

IBM Developer

SKILLS NETWORK



Question

True or False, the following plot shows that as the order of the polynomial increases the mean square error of our model decreases on the test data:

Error

MSE

Training Error

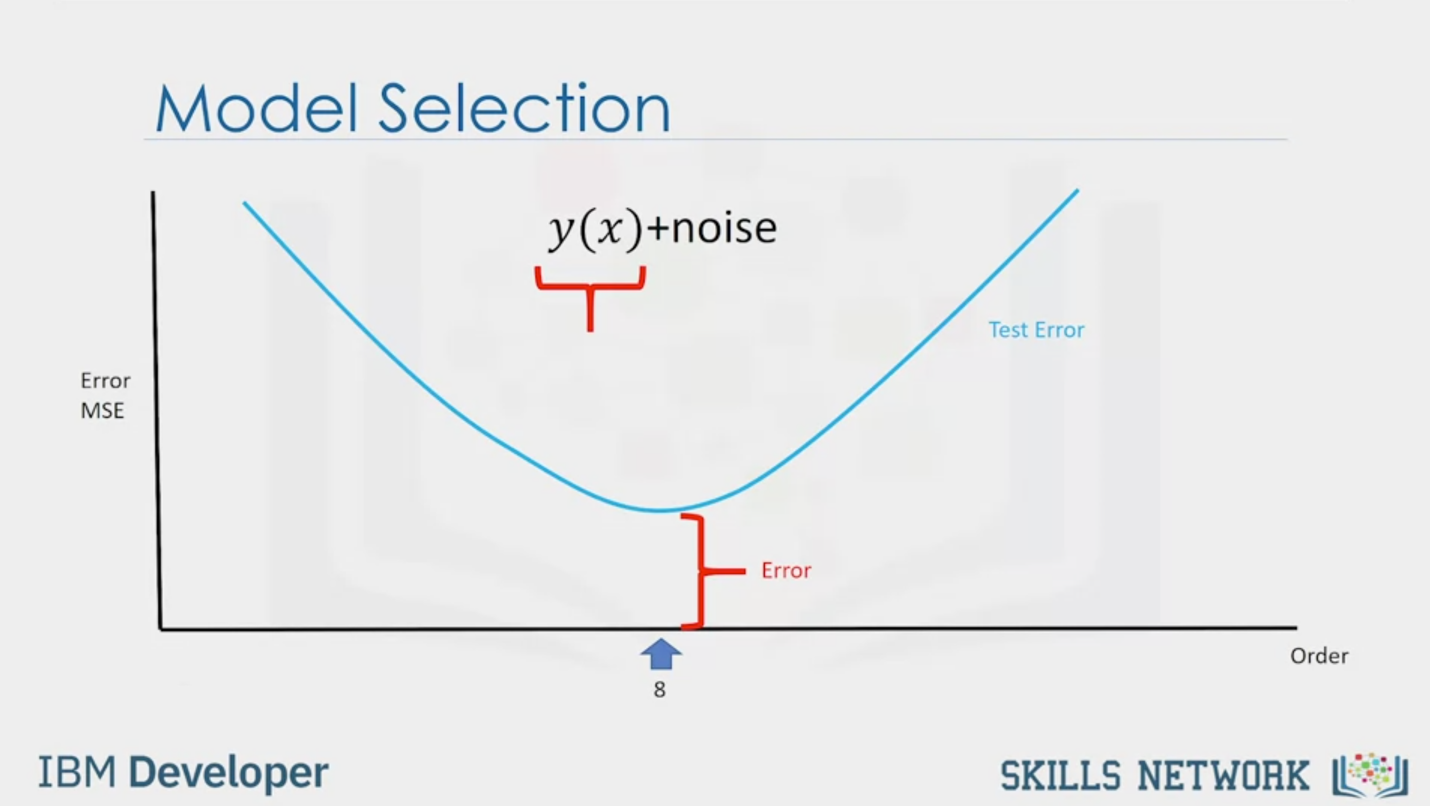
Order

False

True

Skip

Send



Model Selection

y(x)+noise

Error

MSE

Error

Test Error

Order

IBM Developer

SKILLS NETWORK



Model Selection

IBM Developer

SKILLS NETWORK



IBM Developer

SKILLS NETWORK



IBM Developer

SKILLS NETWORK



IBM Developer

SKILLS NETWORK



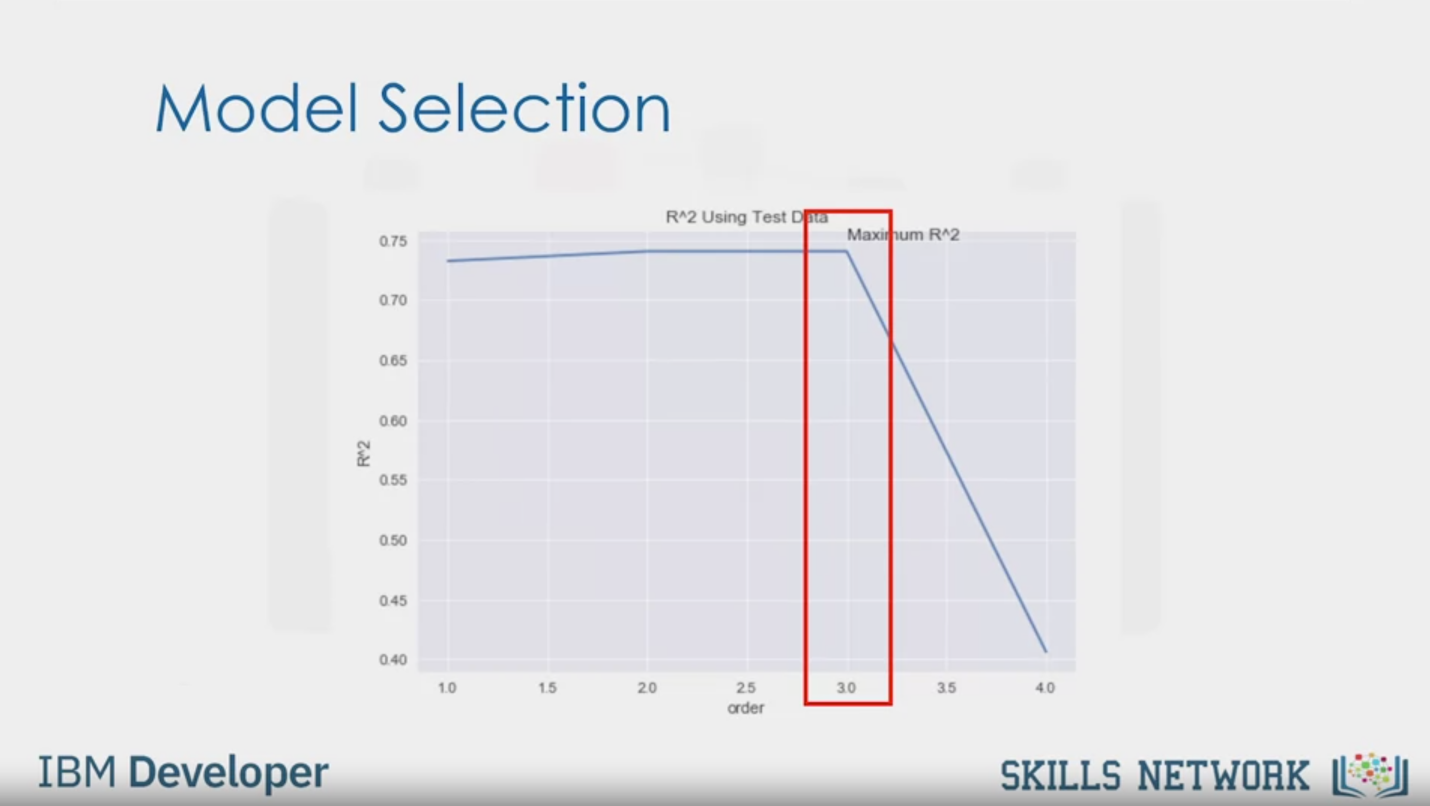
IBM Developer

SKILLS NETWORK



IBM Developer

SKILLS NETWORK



Model Selection

R^2

R^2 Using Test Data

Maximum R^2

order

IBM Developer

SKILLS NETWORK



Rsqu\_test=[]

order=[1,2,3,4]

for n in order:

pr=PolynomialFeatures(degree=n)

x\_train\_pr=pr.fit\_transform(x\_train[['horsepower']])

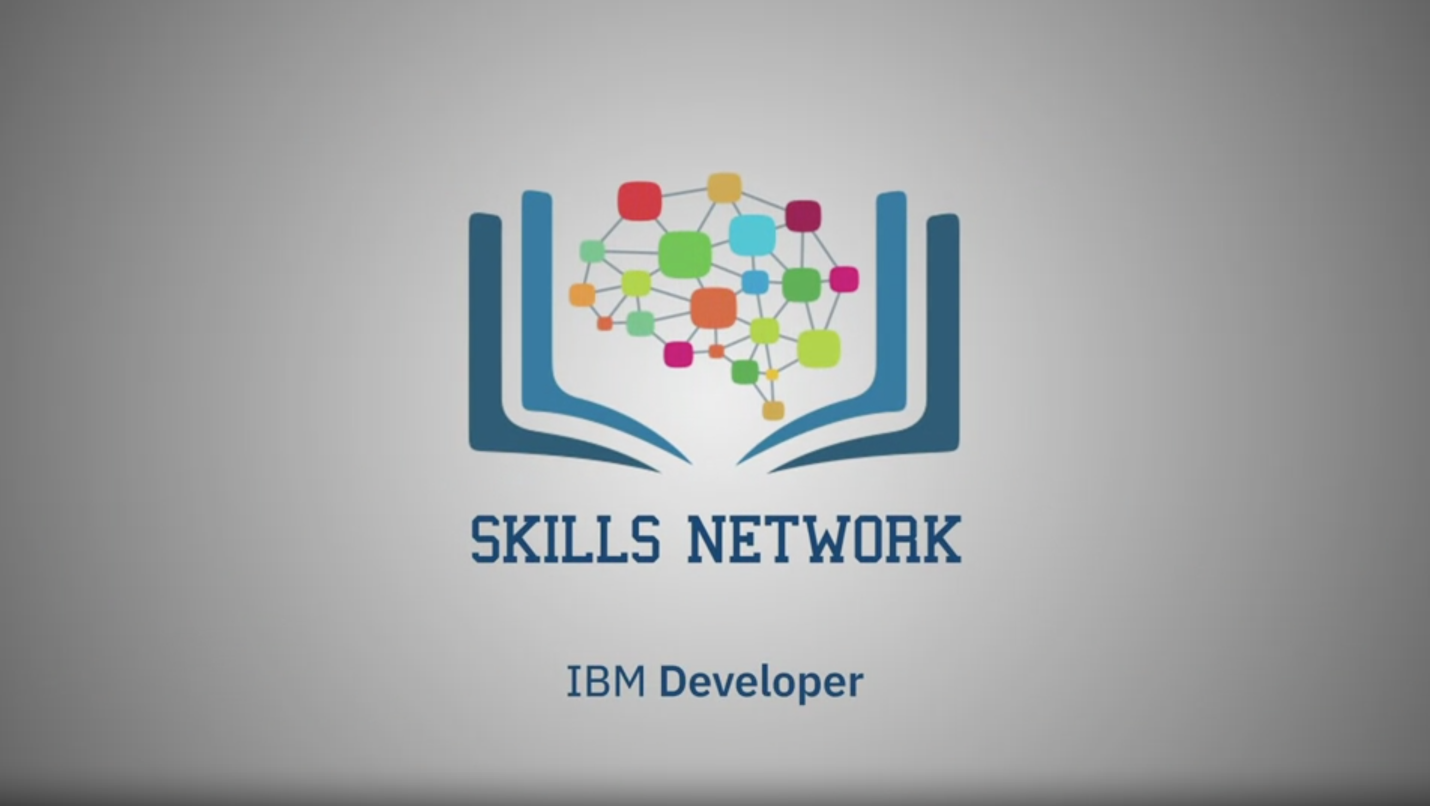
x\_test\_pr=pr.fit\_transform(x\_test[['horsepower"]])

Ir.fit(x\_train\_pr,y\_train)

Rsqu\_test.append(Ir.score(x\_test\_pr,y\_test))

IBM Developer

SKILLS NETWORK



SKILLS NETWORK

IBM Developer

If you recall, in the last module,

we discussed polynomial regression.

In this section, we will discuss how to pick

the best polynomial order and problems

that arise when selecting the wrong order polynomial.

Consider the following function,

we assume the training points come from a polynomial function plus some noise.

The goal of Model Selection is to determine the order of

the polynomial to provide the best estimate of the function y(x).

If we try and fit the function with a linear function,

the line is not complex enough to fit the data.

As a result, there are many errors.

This is called underfitting,

where the model is too simple to fit the data.

If we increase the order of the polynomial,

the model fits better, but the model is

still not flexible enough and exhibits underfitting.

This is an example of the 8th order polynomial used to fit the data.

We see the model does well at fitting the data and estimating

the function even at the inflection points.

Increasing it to a 16th order polynomial,

the model does extremely well at tracking

the training point but performs poorly at estimating the function.

This is especially apparent where there is little training data.

The estimated function oscillates not tracking the function.

This is called overfitting,

where the model is too flexible and fits the noise rather than the function.

Let's look at a plot of the mean square error for

the training and testing set of different order polynomials.

The horizontal axis represents the order of the polynomial.

The vertical axis is the mean square error.

The training error decreases with the order of the polynomial.

The test error is a better means of estimating the error of a polynomial.

The error decreases 'till the best order of the polynomial

is determined. Then the error begins to increase.

We select the order that minimizes the test error.

In this case, it was eight.

Anything on the left would be considered underfitting.

Anything on the right is overfitting.

If we select the best order of the polynomial,

we will still have some errors.

If you recall the original expression for the training points we see a noise term.

This term is one reason for the error.

This is because the noise is random, and we can't predict it.

This is sometimes referred to as an irreducible error.

There are other sources of errors as well.

For example, our polynomial assumption may be wrong.

Our sample points may have come from a different function.

For example, in this plot,

the data is generated from a sine wave.

The polynomial function does not do a good job at fitting the sine wave.

For real data, the model may be too difficult to

fit or we may not have the correct type of data to estimate the function.

Let's try different order polynomials on the real data using horsepower.

The red points represent the training data.

The green points represent the test data.

If we just use the mean of the data,

our model does not perform well.

A linear function does fit the data better.

A second order model looks similar to the linear function.

A third order function also appears to increase,

like the previous two orders.

Here, we see a fourth order polynomial.

At around 200 horsepower,

the predicted price suddenly decreases.

This seems erroneous.

Let's use R-squared to see if our assumption is correct.

The following is a plot of the R-squared value.

The horizontal axis represents the order polynomial models.

The closer the R-squared is to one, the more accurate the model is.

Here, we see the R-squared is optimal when the order of the polynomial is three.

The R-squared drastically decreases when the order is increased to four,

validating our initial assumption.

We can calculate different R-squared values as follows.

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

We create a list containing different polynomial orders.

We then iterate through the list using a loop.

We create a polynomial feature object with the order of the polynomial as a parameter.

We transform the training and test data into a polynomial using the fit transform method.

We fit the regression model using the transform data.

We then calculate the R-squared using the test data and store it in the array.