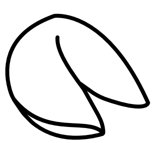
INSEAD The Business School for the World®

Machine Learning and Optimization Lecture 3

Supervised learning: basics & regression recap

Professor Georgina Hall



Agenda for today

Learning objectives:

- Revisit linear regression and regression
- Understand some basic concepts of supervised learning: overfitting, testing/training/validation sets

How will we get there?

- Make use of your prior knowledge about

regression

- Activity: M. Gelato’s ice-creams

Supervised learning

Recall from Lecture 1:

What is the difference
between **supervised** and **unsupervised** learning?

Data with **labels** or not.

→ Notion of what the “right” answer is in supervised

You already know one example of supervised learning:
linear regression

Linear Regression

Before moving on…

Please download

- ML&O Lecture 3 - Exercise_book.ipynb
- women_data.csv
- Gelato_times_sales.csv

Linear regression (1/7)

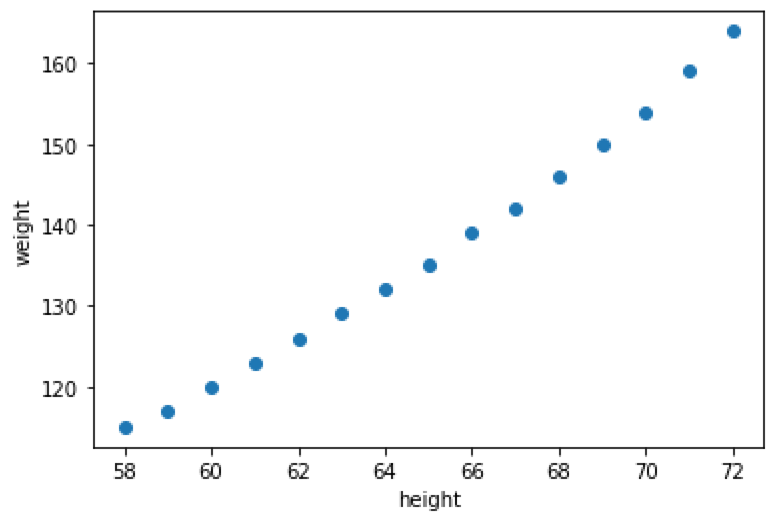
We are going to do regression on an easy dataset: **women_data**

1. Take a look at it. How many observations?
2. Plot weight as a function of height using plt.scatter.
3. Which one is the independent variable? The dependent variable?

```
women_data.head()
```

	height	weight
0	58	115
1	59	117
2	60	120
3	61	123
4	62	126
5	63	129
6	64	132
7	65	135
8	66	139
9	67	142
10	68	146
11	69	150
12	70	154
13	71	159
14	72	164

```
plt.scatter(women_data["height"],women_data["weight"])
plt.xlabel("height")
plt.ylabel("weight")
```



Linear regression (2/7)

- Up until now, we've called **feature** any column/variable in our dataset.
- In regression, there is a special variable/feature: **the dependent variable.**
- We refer to the **dependent variable** as the **label (="special" feature).** The **independent variables** are the **features.**

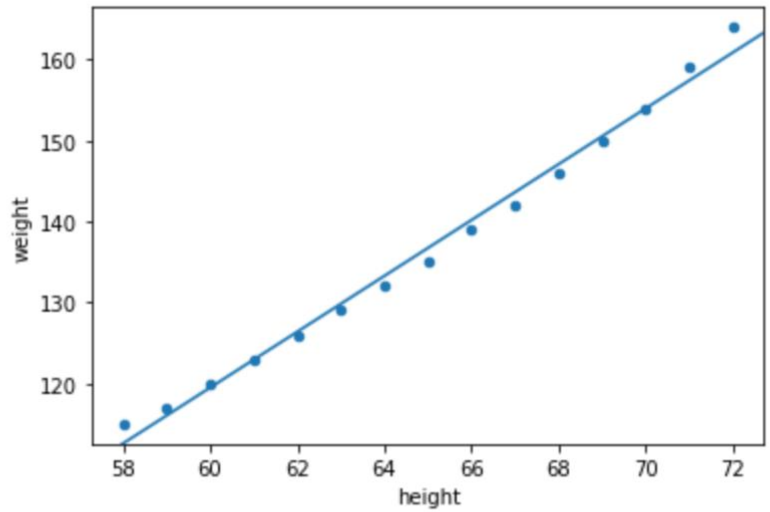
Label here is weight. Feature is height.

Goal: If you are given a height, can you predict a weight?

How to do this? One way: linear regression.

Linear regression (3/7)

In linear regression, we **fit a line to the data**. How do we add this line?



Input: $(height_i, weight_i)$ pairs (15 of them)

Goal: find numbers $(a = intercept, b = slope)$ such that sum of residuals squared $(weight_1 - a - b \cdot height_1)^2 + \cdots + (weight_n - a - b \cdot height_n)^2$ is smallest possible

Output: Once we have (a, b), we can obtain the predicted values $weight_{new} = a + b \cdot height_{new}$

Linear regression (4/7)

How to code this up in Python?

We use the **Scikit package:**

```
from sklearn.linear_model import LinearRegression
```

First part: fitting the model, i.e., fitting the line:

```
X=women_data[["height"]]
Y=women_data[["weight"]]

lm = LinearRegression().fit(X, Y)
```

Specify what your x and y variables are

Model you’re interested in

Fits that model to your data

Linear regression (5/7)

Second part: getting relevant parameters outputted

```
print("Intercept = ",lm.intercept_) # Print the resultant model intercept

print("Model coefficients = ", lm.coef_) # Print the resultant model coefficients (in order of variables in X)

print("R^2 =",lm.score(X,Y)) # Print the resultant model R-squared
```

See everything you can get in the documentation:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Third part: getting the predictions

```
Y_pred=lm.predict(X)
```

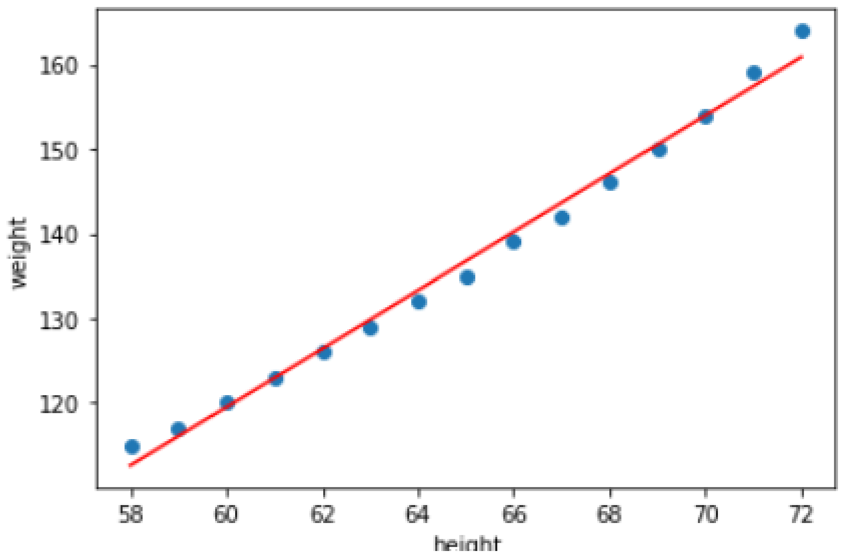
Model is named “lm” and we predict from that model

How are we getting these predictions? If we had a new datapoint, e.g. 62.5, how would it work to get a prediction?

Linear regression (6/7)

Fourth part (Optional): Plotting

How can we use plt.plot() and plt.scatter() to get the graph below?




```
plt.scatter(X,Y)
plt.plot(X,Y_pred,c="red")
plt.xlabel("height")
plt.ylabel("weight")
```

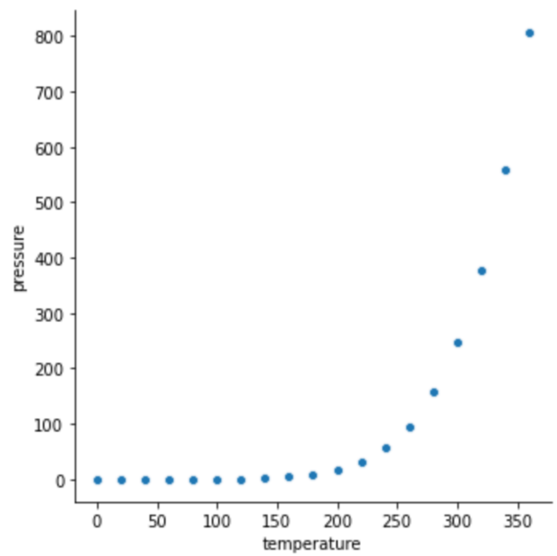
Linear regression (7/7)

Why do we do linear regression?

The goal is to predict new values.

For example, if an American woman ages 30-39 gives us her height, we should be able to infer her weight by using our regression line.

Linear regression is not always good enough….

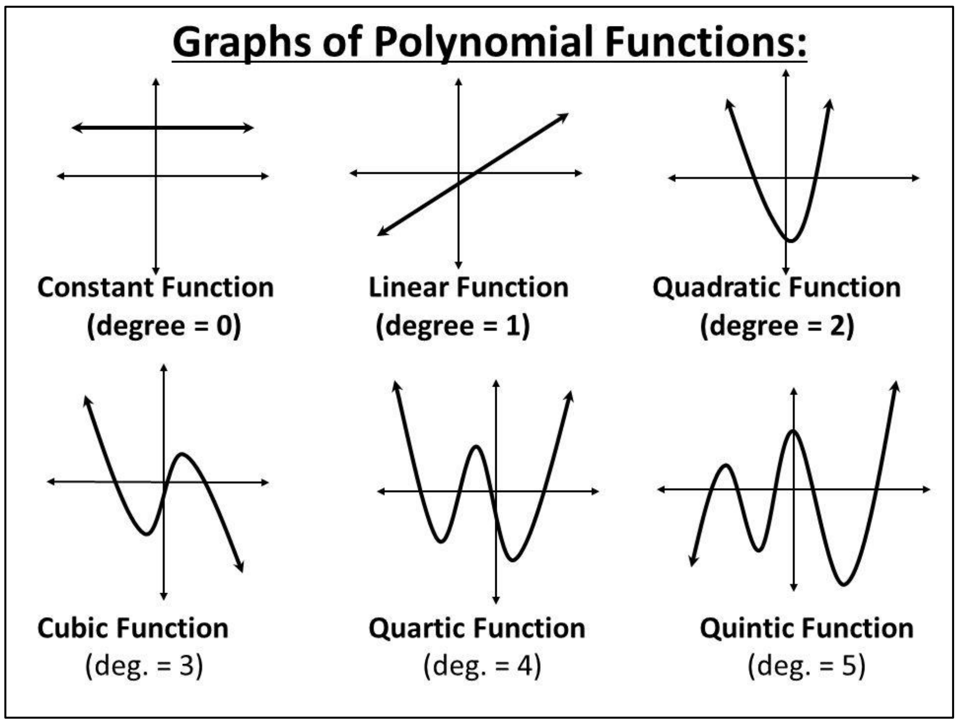


Polynomial Regression

Regression in general

- In (general) regression, our goal is to fit a **curve** to data (not just a line as in linear regression).
- Of particular interest to us now is going to be **polynomial regression.**

Why? It is “easy” to do like linear regression but is more versatile.



Source:http://brandon.ai/

Polynomial Regression (1/5)

We use scikit-learn again for this, but it is a bit more complicated.

Let's take a look at the code together!

First part: specify the degree of interest and the features and label.

```
degree=3
X=women_data[["height"]]
Y=women_data[["weight"]]
```

Polynomial Regression (2/5)

Second step: go from the datapoints to their "polynomial version"

For example, height datapoint 58 becomes $[1, 58, 58^2, 58^3]$

Intercept

Powers up to 3 as degree=3

```
poly = PolynomialFeatures(degree) #define the polynomial
X_poly=poly.fit_transform(X) #map all the values of X as [1,x,x^2,x^3, etc]
```

Your turn!
Double-check that this is the case by taking a look at X and X_poly.

```
[1.00000e+00, 5.80000e+01, 3.36400e+03, 1.95112e+05]
```

=1 =58 $= 58^2$ $= 58^3$

Polynomial Regression (3/5)

Third step: fit a Linear Regression model to the polynomial datapoints.

```
polyreg = LinearRegression().fit(X_poly, Y)
```

Why linear regression?

We want to find the **coefficients** of the polynomial:

$$c_0 + c_1 x + c_2 x^2 + c_3 x^3$$

Now that we've made the datapoints polynomial, the expression above is **linear** in the coefficients ⇒ **Linear Regression**

Polynomial Regression (4/5)

Fourth step: get the coefficients (c_0, c_1, c_2, c_3) and R^2.

Your turn!

```
print(polyreg.coef_) #print these coefficients
print(polyreg.score(X_poly,Y)) #print R^2
```

```
[[ 0.00000000e+00  4.64107891e+01 -7.46184371e-01  4.25255572e-03]]
0.9997816939979363
```

Fifth step: Predict new points

```
y_pred=polyreg.predict(X_poly)
```

Same as before, except that we're applying it to X_poly, the transformed variables.

Polynomial Regression (5/5)

Sixth step: plot! A bit more complicated than for linear regression:

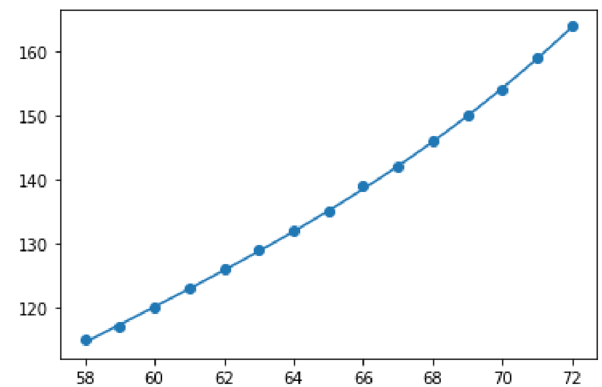
- A line is defined by two points but not a polynomial
- We need many more points to be able to generate a “pretty” curve.

```
linepoints = np.linspace(np.min(X), np.max(X), 100)
linepoints_poly=poly.fit_transform(linepoints)
linepoints_pred=polyreg.predict(linepoints_poly)

plt.plot(linepoints,linepoints_pred)
```

Here 100 points between 58 and 72!

Your turn! Try running this code!



Discovering the basic concepts of supervised learning

Let’s start with an activity in BORs.

Selling ice-creams

M. Gelato is the owner of an ice-cream parlor.

- Over the course of the year, he has written down some of his **daily sales**.
- He knows that his sales are **periodic**: i.e., every year, he sells in approximately the same way
- He would like to fit a curve to his data so as to better predict what his sales are going to be next year.

Goal: help M. Gelato!

In BORs (LG 1-15) with your groups: complete part 4 of the Notebook.

Slides are on the course website.

Activity wrap-up (1/3)

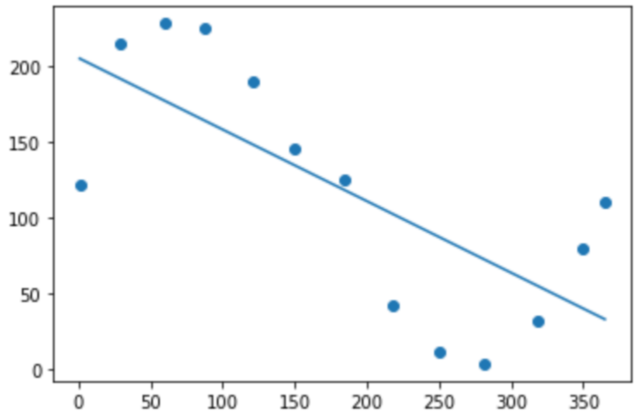
```
Gelato=pd.read_csv("Gelato_Times_Sales.csv")
Gelato
```

	Times	Sales
0	1	122
1	29	215
2	60	228
3	88	225
4	121	190
5	150	145
6	184	125
7	218	43
8	250	12
9	281	4
10	318	32
11	350	80
12	365	110

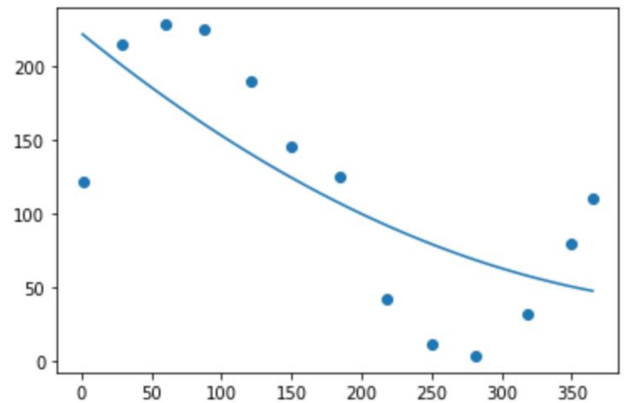
Why do we normalize the data here?

We build $[1, x, x^2, \ldots, x^{12}]$ in the worst case and 365^{12} is huge! If we normalize, we don't have as huge numbers.

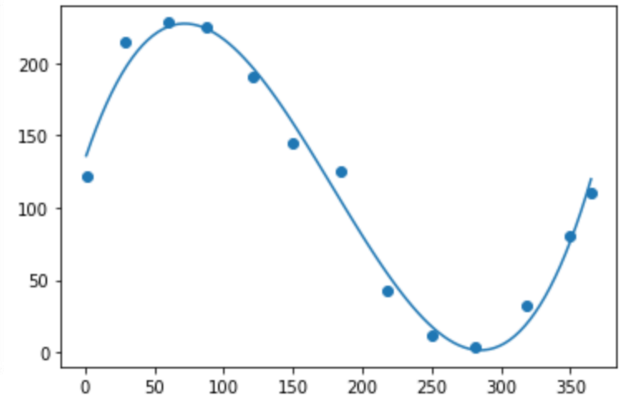
Activity wrap-up (2/3)



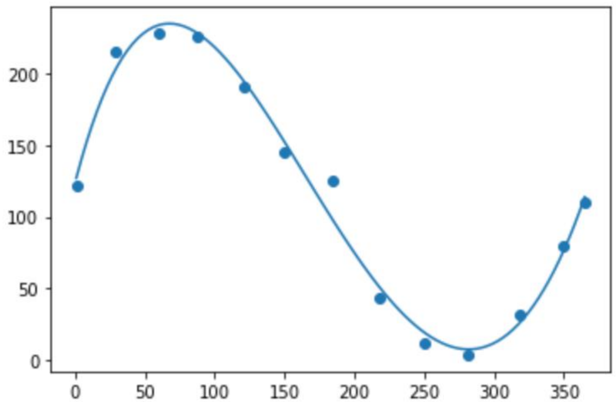
d=1, R^2=0.516



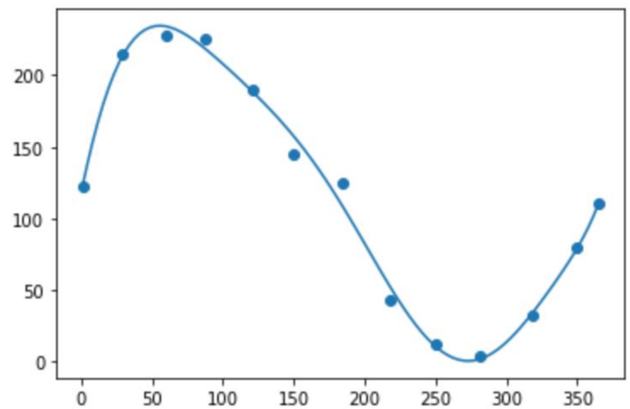
d=2, R^2=0.531



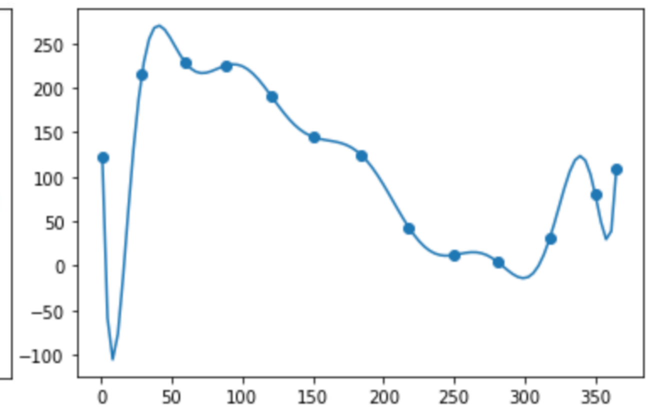
d=3, R^2=0.979



d=5, R^2=0.985



d=8, R^2=0.992



d=12, R^2=1

Activity wrap-up (3/3)

- **R^2 increases** as the degree increases as there is more and more “freedom” in how the polynomial curve can twist.
- **The fit gets stranger and stranger** (negative values when d=12…)

Conclusions:

- This is known as **overfitting:** regression curve fits “too closely” to existing datapoints.
- Ends up **not reflecting reality** as too tailored to the dataset we have: poor prediction abilities
- Need to find **new ways of measuring** what a “good fit” is!

How to measure how good a model is at prediction?

How good is a model at predicting? (1/3)

If we only use the data at hand to evaluate our model, then the model can **overfit to the data**.

What if I gave you additional datapoints? Could we now evaluate how good the model is at predicting?

Time	Sales
10	146.4116
35	195.8377
70	241.7297
135	216.1947
163	166.1761
192	104.6491
228	36.80205
302	14.49156

Yes!

Idea: See how good the model is at predicting sales for these new points by comparing to "real" sales.

⇒ Tells you how good the model is when faced with points it's never seen.

How good is a model at predicting? (2/3)

We predict the sales for these new time points using our **linear regression model.**

Time	Sales
10	146.4116
35	195.8377
70	241.7297
135	216.1947
163	166.1761
192	104.6491
228	36.80205
302	14.49156

Root Mean Squared Error (RMSE) $= \sqrt{\frac{1}{n} \cdot \left((pred_1 - actual_1)^2 + \cdots + (pred_n - actual_n)^2\right)}$

How good is a model at predicting? (3/3)

We can do this for each model we looked at.

Degree	RMSE
1	51.18
2	54.5
3	18.6
5	21.33
8	20.86
12	93.26

- The best model from this table is degree =3
- This ties in with what our intuition would have had us pick!

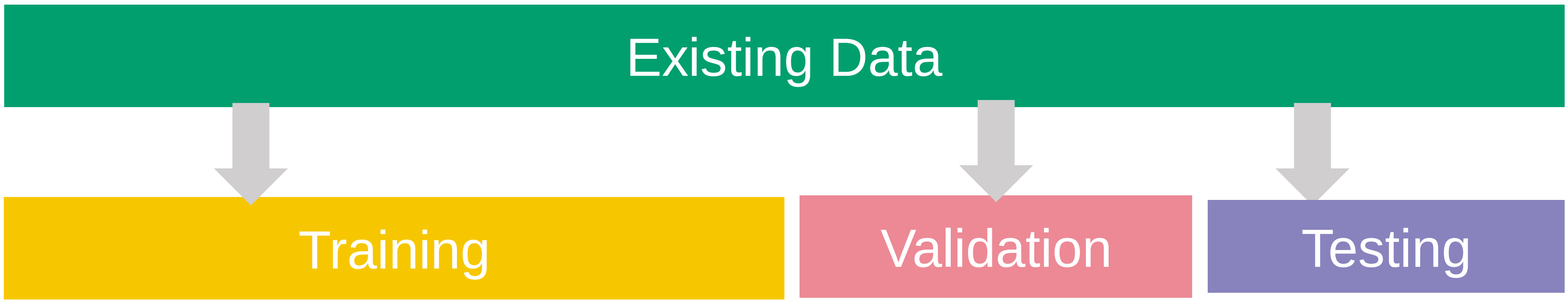
Here, I gave you new datapoints for this to work. But how would we proceed if we didn't have new datapoints?

Idea: When you first get your dataset, split it up and only build your model on part of it! Use the other part to evaluate it.

Training, validation, and test sets

Training, validation, and test sets (1/2)

We in fact divide the existing data into **three.**

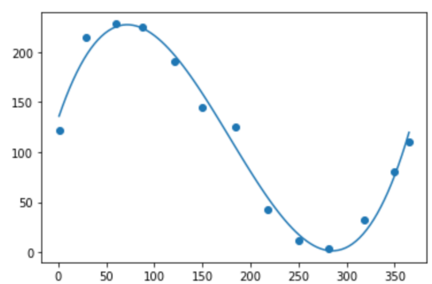


- The **training set** serves to **build your model**.
- The **validation set** serves to **select between models**.
- The **test set** is used just once to **give an indication as to how the chosen model will perform.**

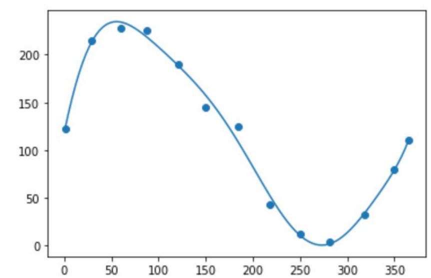
Training, validation, and test sets (2/2)

Example: Polynomial regression

1. Use the **training set** to come up with polynomial regressors with different degrees.



d=3



d=8

2. Use the **validation set** to pick which degree is the best, e.g. $d = 3$.

3. Use the **testing set** to evaluate how well the model you picked would perform on new data.

These concepts in Python

We use scikit again.

- How to **divide** a dataset into **two**:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( Xdata, ydata, test_size=0.33)
```

- How to **divide** a dataset into **three**: use the function above twice (see homework)
- How to **compute the RMSE** between real values and predicted values (example here with the linear regression model)

```
y_pred_val=lm.predict(Xval)
from sklearn.metrics import mean_squared_error
mean_squared_error(Yval,y_pred_val)**(1/2)
```

Supervised learning process

Data

A **model** with p **hyperparameters (Ex: degree)**

Separate data into **training, validation, testing sets**

Do the **same data pre-processing** on each set **separately**

Train the model on **training set varying** all values of hyperparameters

Validate the hyperparameters on validation set
& retrain on training + validation

Test on testing set

If small error, communicate model with validated hyperparameters and test error

If large error, bin everything and start over, changing the model.

- Ideally: have different validation sets for each parameter so as not to **overfit the validation set**
- **Requires a lot of data /computation power**

Wrap-up & Next time

Today, we:

- **Reviewed linear** and **polynomial regression**
- Understood the concept of **overfitting**
- Understood the purpose of **training/testing/validation sets**
- Saw **how to approach** supervised learning problems such as regression

Next time:

- Pitfalls of Machine Learning + Starting on Classification
- Mini-quiz to do + homework

INSEAD

The Business School for the World®

EUROPE | ASIA | MIDDLE EAST | NORTH AMERICA