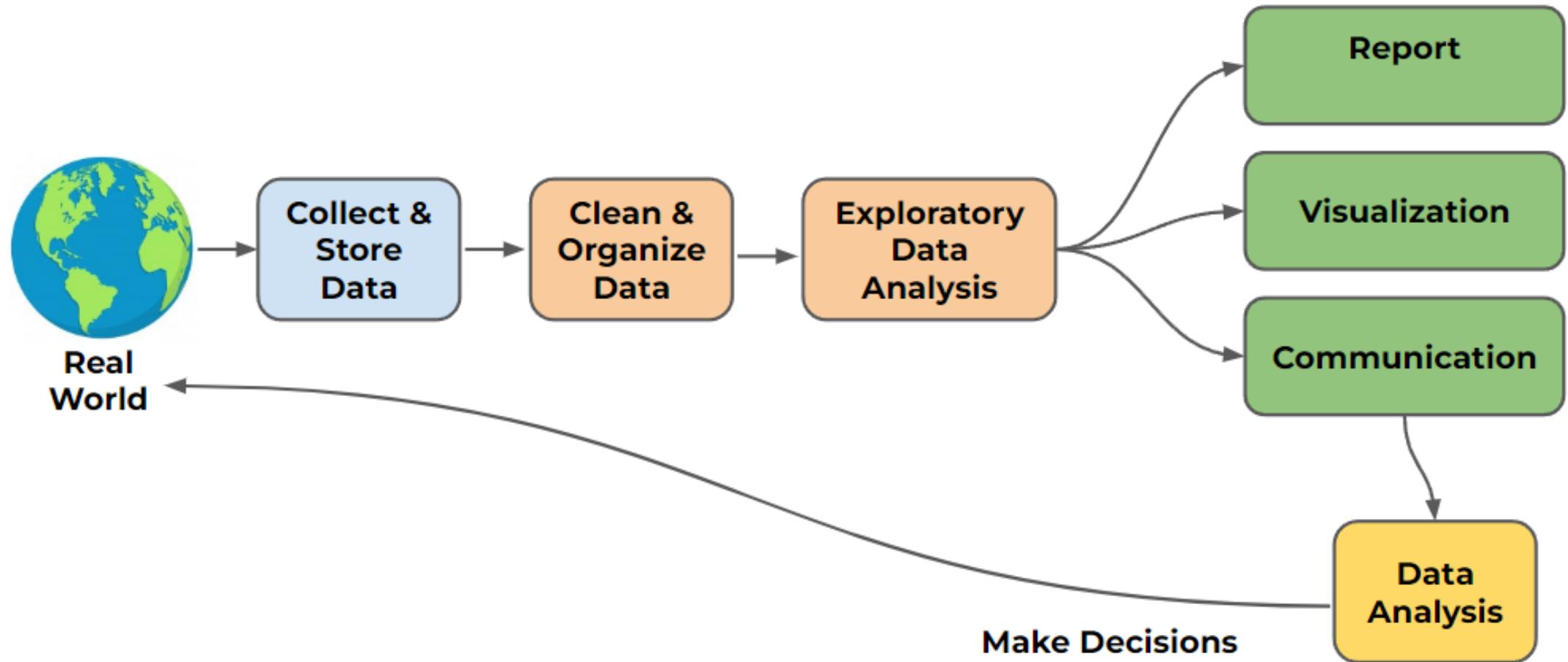
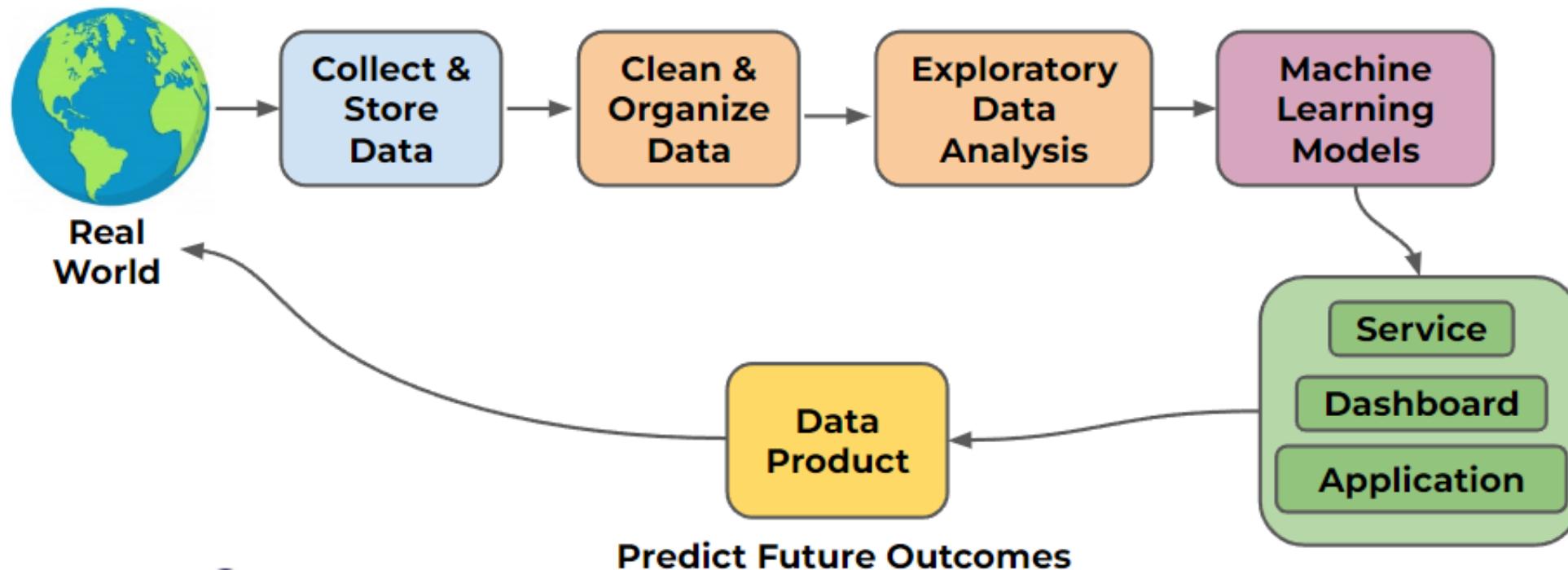


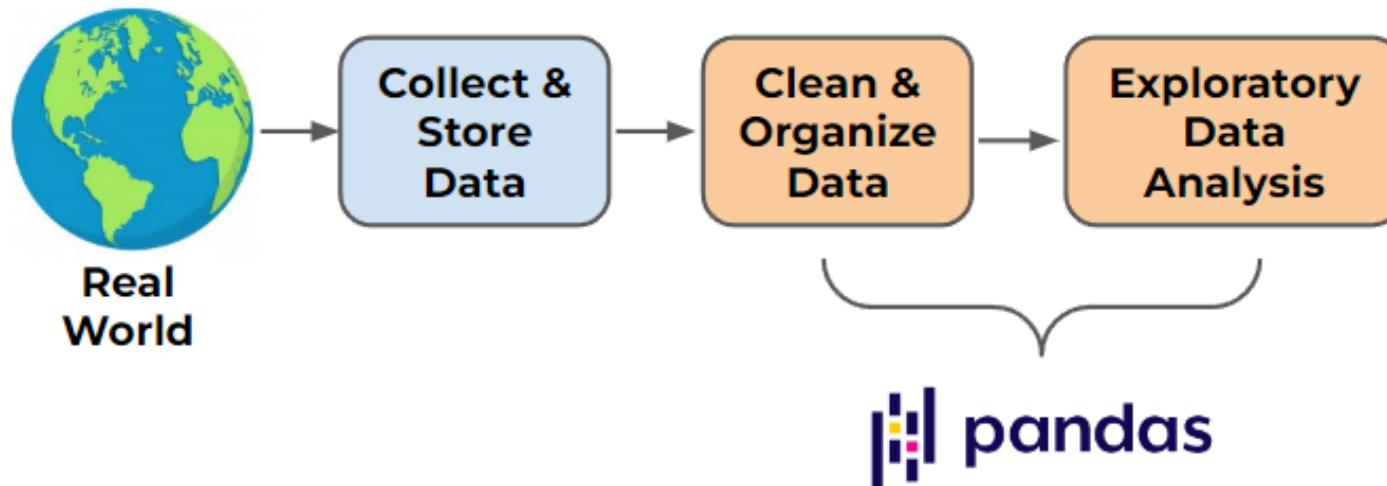
Applied AI From Scratch Using Python





- What is NumPy?
 - Python library for creating N-dimensional arrays
 - Ability to quickly broadcast functions
 - Built-in linear algebra, statistical distributions, trigonometric, and random number capabilities

- Why use NumPy?
 - While NumPy structures look similar to standard Python lists, they are **much** more efficient!
 - The broadcasting capabilities are also extremely useful for quickly applying functions to our data sets.



- Pandas is a library for Data Analysis.
- Extremely powerful table (DataFrame) system built off of NumPy.
- Fantastic documentation:
 - <https://pandas.pydata.org/docs/>



- What can we do with Pandas?
 - Tools for reading and writing data between many formats.
 - Intelligently grab data based on indexing, logic, subsetting, and more.
 - Handle missing data.
 - Adjust and restructure data.

- Series and DataFrames
- Conditional Filtering and Useful Methods
- Missing Data
- Group By Operations
- Combining DataFrames
- Text Methods and Time Methods
- Inputs and Outputs



Real
World

Collect &
Store
Data

Clean &
Organize
Data

Exploratory
Data
Analysis

Machine
Learning
Models

Supervised Learning:
Predict an Outcome
Unsupervised Learning:
Discover Patterns in Data

- Our main goals in ML Overview section:
 - Problems solved by Machine Learning
 - Types of Machine Learning
 - Supervised Learning
 - Unsupervised Learning
 - ML Process for Supervised Learning

- Many other relevant topics will be discussed later in the course as we “discover” them, including:
 - Bias-Variance Trade-off
 - Cross-validation
 - Feature Engineering
 - Scikit-learn
 - Performance Metrics and much more!

- Machine learning in general is the study of statistical computer algorithms that improve automatically through data.
- This means unlike typical computer algorithms that rely on human input for what approach to take, ML algorithms infer best approach from the data itself.

- Machine learning is a subset of Artificial Intelligence.
- ML algorithms are not explicitly programmed on which decisions to make.
- Instead the algorithm is designed to infer from the data the most optimal choices to make.

- What kinds of problems can ML solve?
 - Credit Scoring
 - Insurance Risk
 - Price Forecasting
 - Spam Filtering
 - Customer Segmentation
 - Much more!

- Structure of ML Problem framing:
 - Given **features** from a data set **obtain** a desired **label**.
 - ML algorithms are often called “estimators” since they are estimating the desired **label** or output.

- How can ML be so robust in solving all sorts of problems?
- Machine learning algorithms rely on data and a set of statistical methods to learn what features are important in data.

- Simple Example:
 - Predict the price a house should sell at given its current features
(Area,Bedrooms,Bathrooms,etc...)

- House Price Prediction
 - Typical Algorithm
 - Human user defines an algorithm to manually set values of importance for each feature.

- House Price Prediction
 - ML Algorithm
 - Algorithm automatically determines importance of each feature from existing data

- Why machine learning?
 - Many complex problems are only solvable with machine learning techniques.
 - Problems such as spam email or handwriting identification require ML for an effective solution.

- Why not just use machine learning for everything?
 - Major caveat to effective ML is good data.
 - Majority of development time is spent cleaning and organizing data, **not** implementing ML algorithms.

- There are two main types of Machine Learning we will cover in upcoming sections:
 - Supervised Learning
 - Unsupervised Learning

- Supervised Learning
 - Using **historical** and **labeled** data, the machine learning model predicts a value.
- Unsupervised Learning
 - Applied to **unlabeled** data, the machine learning model discovers possible patterns in the data.

- Supervised Learning
 - Requires **historical labeled** data:
 - Historical
 - Known results and data from the past.
 - Labeled
 - The desired output is known.

- Supervised Learning
 - Two main label types
 - Categorical Value to Predict
 - Classification Task
 - Continuous Value to Predict
 - Regression Task

- Supervised Learning
 - Classification Tasks
 - Predict an assigned category
 - Cancerous vs. Benign Tumor
 - Fulfillment vs. Credit Default
 - Assigning Image Category
 - Handwriting Recognition

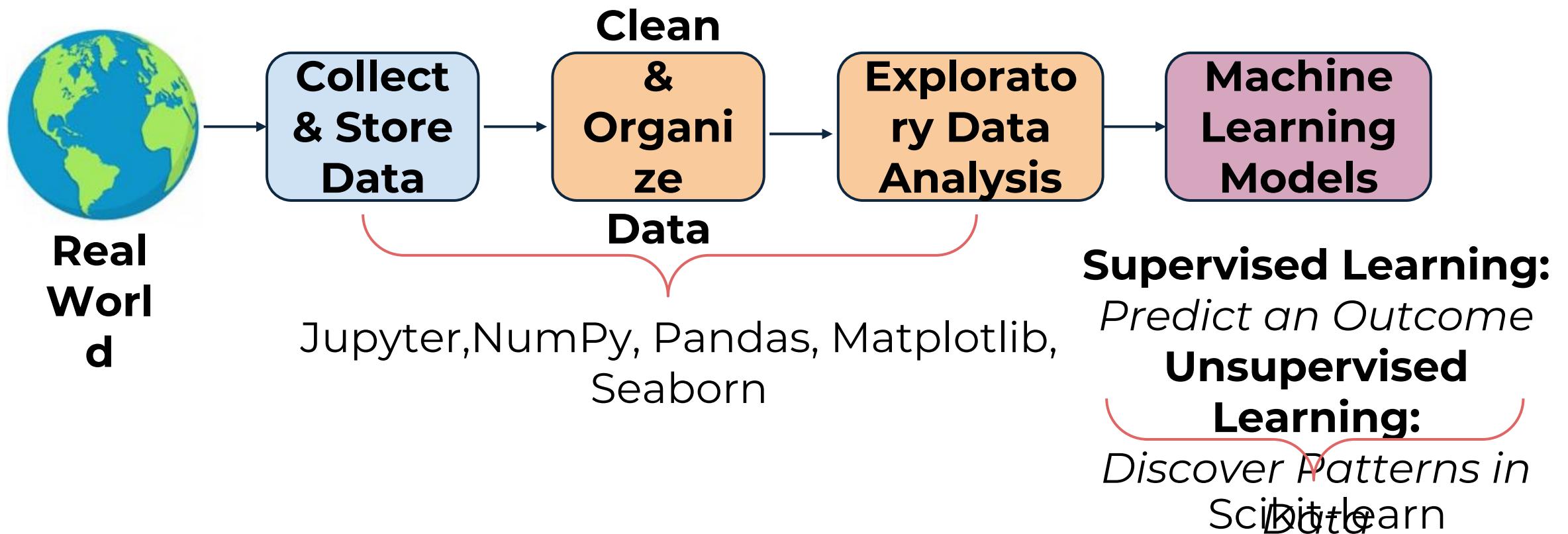
- Supervised Learning
 - Regression Tasks
 - Predict a continuous value
 - Future prices
 - Electricity loads
 - Test scores

- Unsupervised Learning
 - Group and interpret data without a label.
 - Example:
 - Clustering customers into separate groups based off their behaviour features.

- Unsupervised Learning
 - Major downside is because there was no historical “correct” label, it is much harder to evaluate performance of an unsupervised learning algorithm.

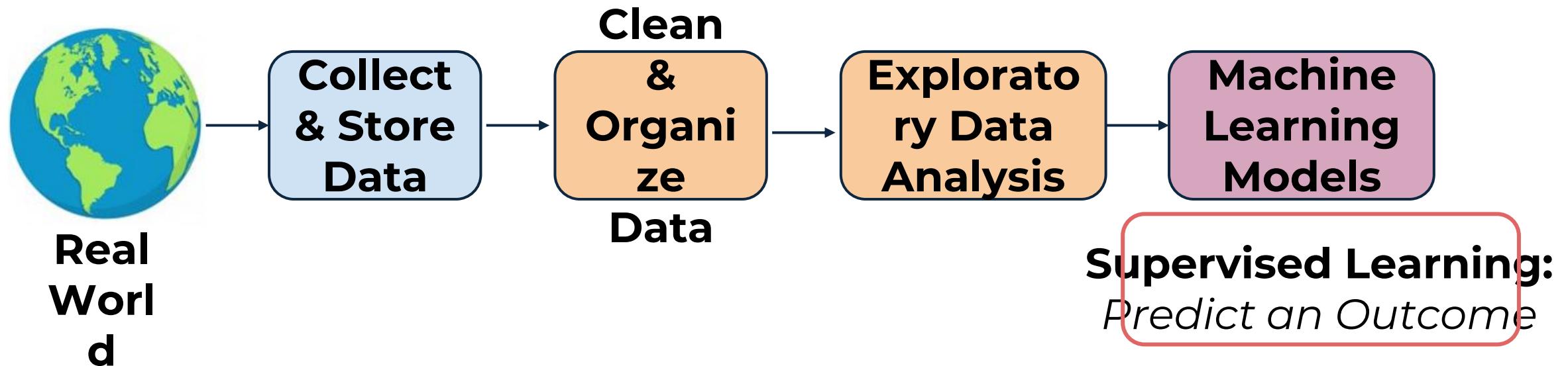
- Machine Learning Sections
 - We first focus on supervised learning to build an understanding of machine learning capabilities.
 - Then shift focus to unsupervised learning for clustering and dimensionality reduction.

- Machine Learning Pathway



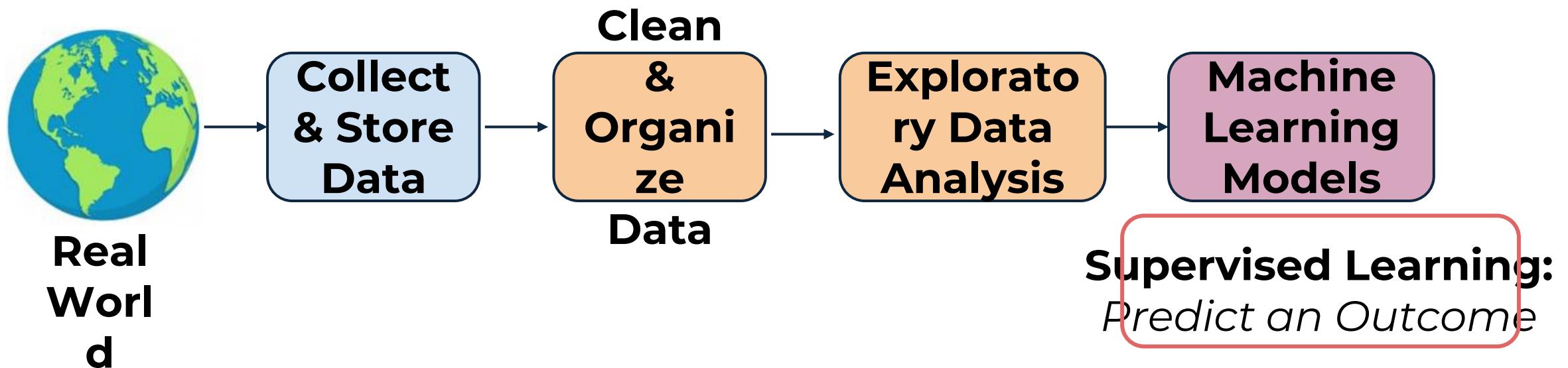
Machine Learning

- Machine Learning Pathway



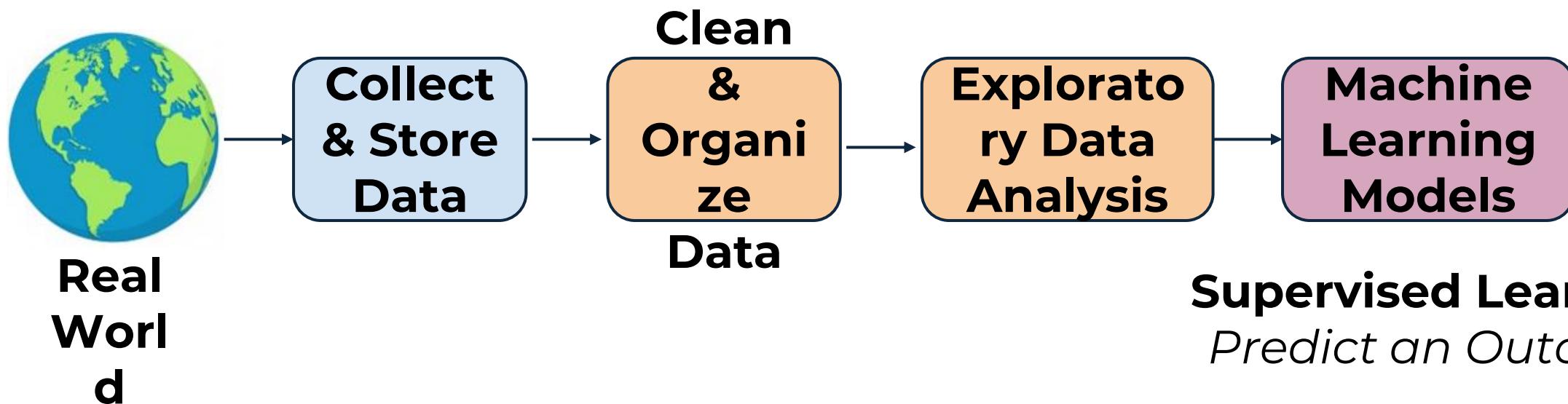
Machine Learning

- ML Process : Supervised Learning Tasks



Machine Learning

- Predict price a house should sell at.



Machine Learning

- **Supervised** Machine Learning Process
- Start with collecting and organizing a data set based on history:

Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000

Machine Learning

- **Historical labeled** data on previously sold houses.

Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000

Machine Learning

- If a new house comes on the market with a known Area, Bedrooms, and Bathrooms:
Predict what price should it sell at.

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000

Machine Learning

- Data Product:
 - Input house features
 - Output predicted selling price

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000

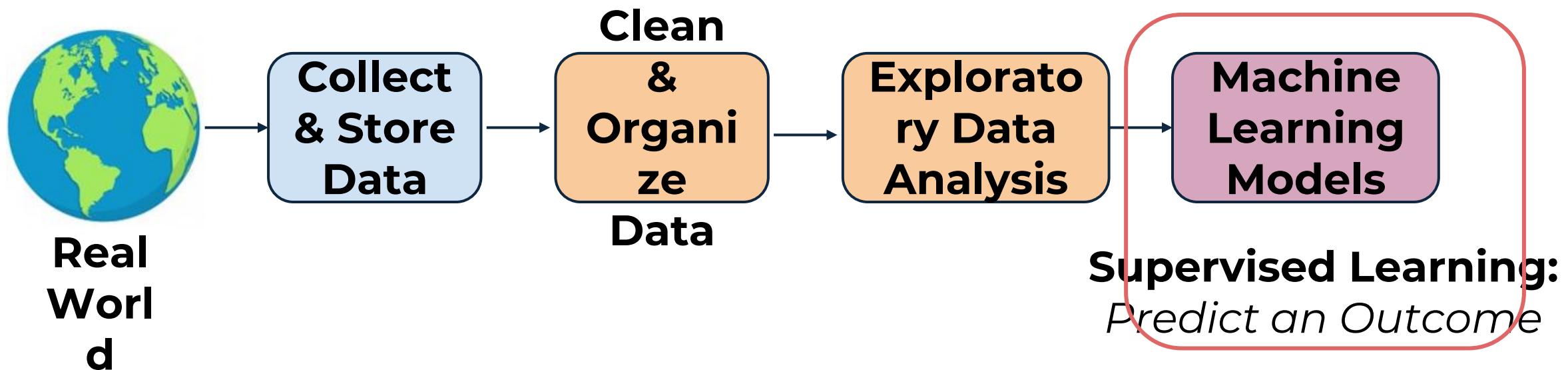
Machine Learning

- Using **historical, labeled** data predict a future outcome or result.

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000

Machine Learning

- Predict price a house should sell at.



Machine Learning

- Predict price a house should sell at.

Machine
Learning
Models

Supervised Learning:
Predict an Outcome

Machine Learning

- **Predict price a house should sell at.**

Machine Learning Models

Supervised Learning:
Predict an Outcome

Machine Learning

- **Predict price a house should sell at.**

Machine Learning Models

Supervised Learning:
Predict an Outcome

Data

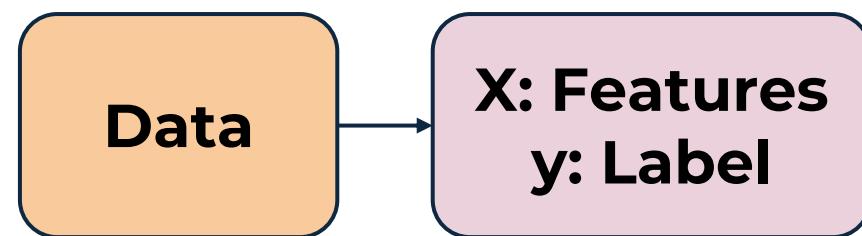
Machine Learning

- **Supervised** Machine Learning Process

Data

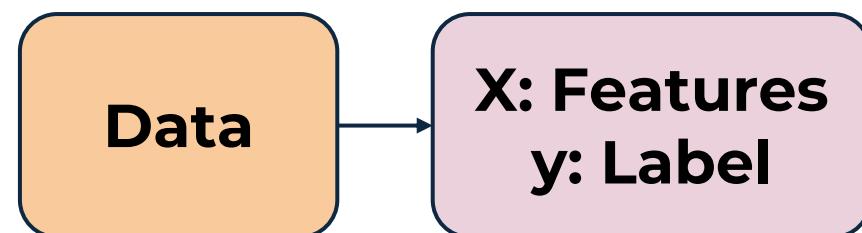
Machine Learning

- **Supervised** Machine Learning Process



Machine Learning

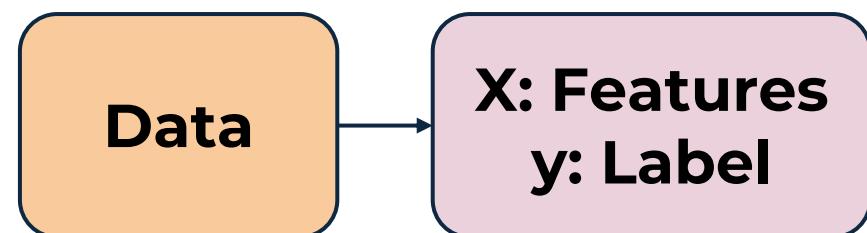
- **Supervised** Machine Learning Process



Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Machine Learning

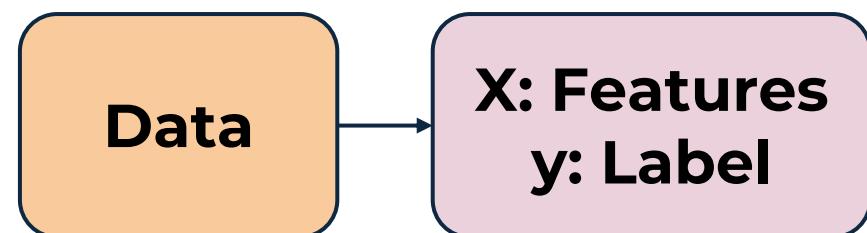
- **Label** is what we are trying to predict



Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Machine Learning

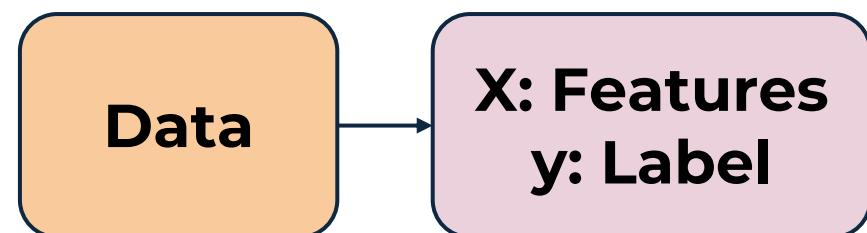
- **Label** is what we are trying to predict



Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Machine Learning

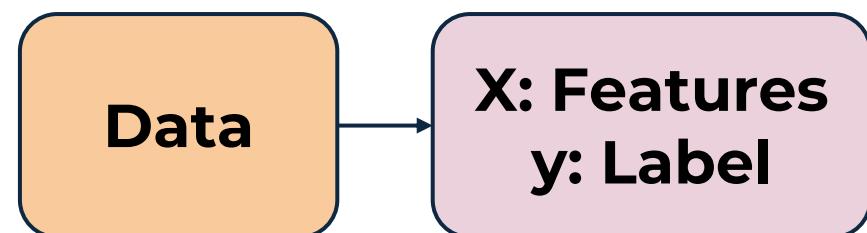
- **Features** are known characteristics or components in the data



Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Machine Learning

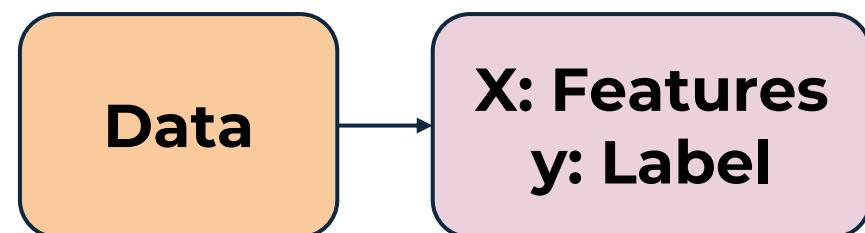
- **Features** are known characteristics or components in the data



Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Machine Learning

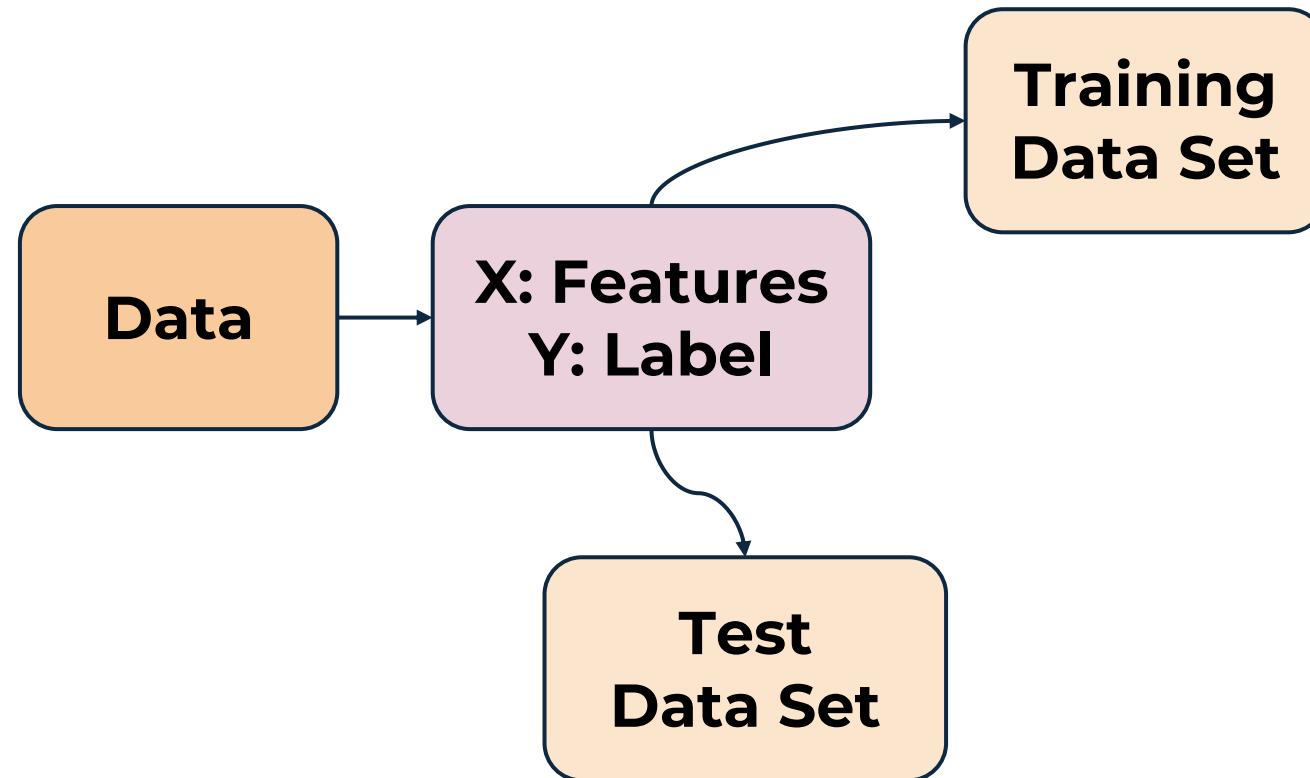
- **Features** and **Label** are identified according to the problem being solved.



Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

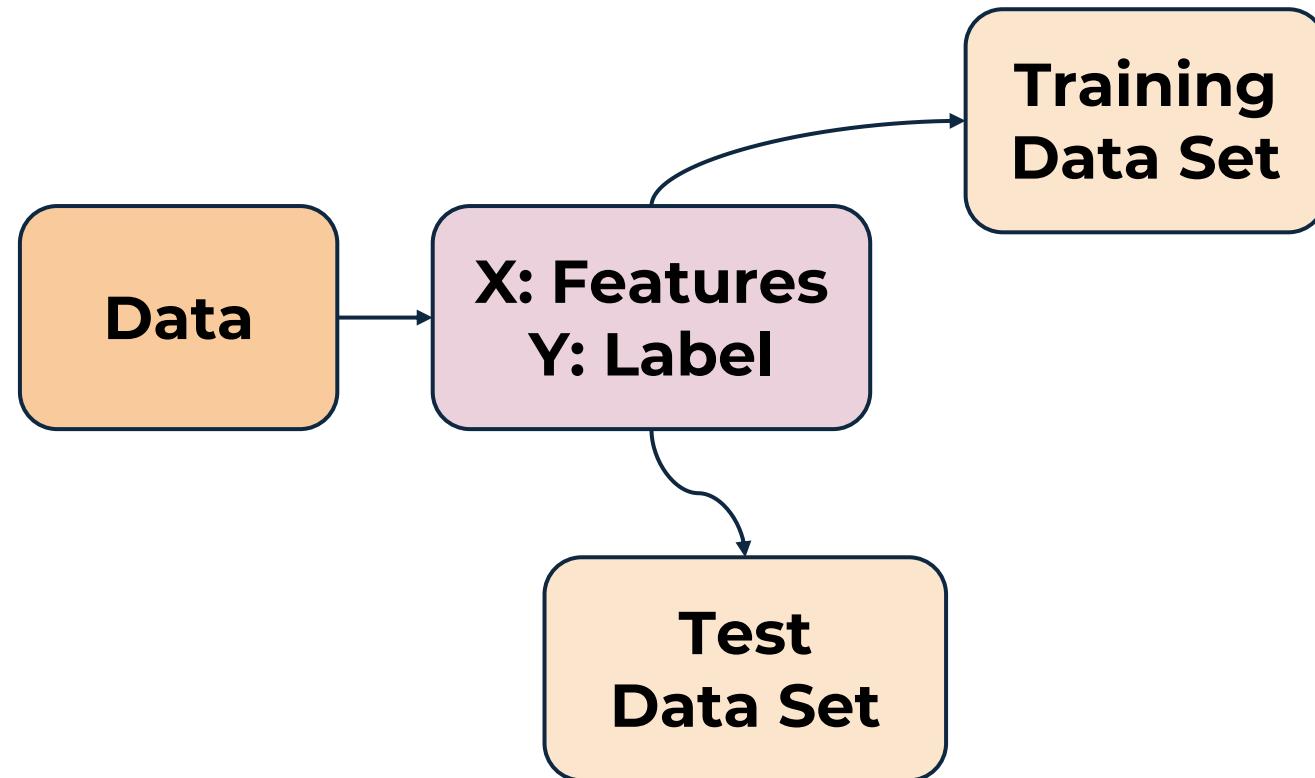
Supervised Machine Learning Process

- Split data into training set and test set



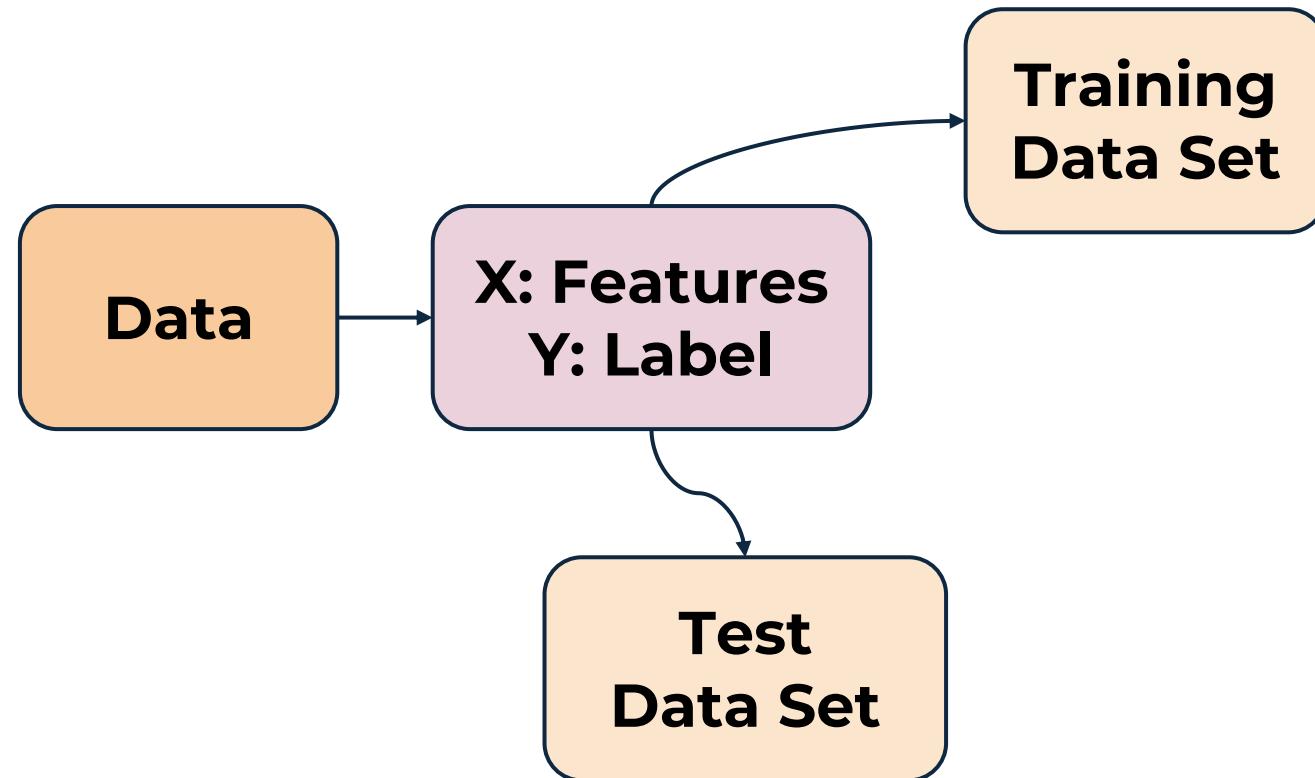
Supervised Machine Learning Process

- Later on we will discuss cross-validation



Supervised Machine Learning Process

- Why perform this split? How to split?



Supervised Machine Learning Process

- Why perform this split? How to split?

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- How would you judge a human realtor's performance?

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- Ask a human realtor to take a look at historical data...

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- Then give her the features of a house and ask her to predict a selling price.

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- But how would you measure how accurate her prediction is? What house should you choose to test on?

Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- You can't judge her based on a new house that hasn't sold yet, you don't know it's true selling price!

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- You shouldn't judge her on data she's already seen, she could have **memorized** it!

Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- Thus the need for a Train/Test split of the data, let's explore further...

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- We already organized the data into Features (X) and a Label (y)

Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- Now we will split this into a training set and a test set:

TRAIN

Area m ²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- Now we will split this into a training set and a test set:

	Area m ²	Bedroom	Bathroo	Price
TRAIN		s	ms	
TEST				
	200	3	2	\$500,000
	190	2	1	\$450,000
	230	3	3	\$650,000
	180	1	1	\$400,000
	210	2	2	\$550,000

Supervised Machine Learning Process

- Notice how we have 4 components

	Area m ²	Bedroom	Bathroo	Price
X TRAIN		s	ms	
200	3	2	\$500,000	
190	2	1	\$450,000	
X TEST				
230	3	3	\$650,000	
180	1	1	\$400,000	
210	2	2	\$550,000	
Y TRAIN				

Supervised Machine Learning Process

- Let's go back to fairly testing our human realtor....

Area m²	Bedroom s	Bathroo ms	Price
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000
180	1	1	\$400,000
210	2	2	\$550,000

Supervised Machine Learning Process

- Let's go back to fairly testing our human realtor....

	Area m ²	Bedroom	Bathroo	Price
TRAIN		s	ms	
	200	3	2	\$500,000
	190	2	1	\$450,000
TEST	230	3	3	\$650,000
	180	1	1	\$400,000
	210	2	2	\$550,000

Supervised Machine Learning Process

- Let her study and learn on the training set getting access to both X and y.

TRAIN

Area m ²	Bedroom	Bathroo	Price
	s	ms	
200	3	2	\$500,000
190	2	1	\$450,000
230	3	3	\$650,000

Supervised Machine Learning Process

- After she has “learned” about the data, we can test her skill on the test set.

TEST

Area m ²	Bedroom	Bathroo
	s	ms
180	1	1
210	2	2

Supervised Machine Learning Process

- Provide only the X test data and ask for her predictions for the sell price.

TEST

Area m ²	Bedroom	Bathroo
	s	ms
180	1	1
210	2	2

Supervised Machine Learning Process

- This is new data she has never seen before!
She has also never seen the real sold price.

TEST

Area m ²	Bedroom	Bathroo
	s	ms
180	1	1
210	2	2

Supervised Machine Learning Process

- Ask for predictions per data point.

Predictions	Area m²	Bedroom s	Bathroo ms
\$410,000	180	1	1
\$540,000	210	2	2

Supervised Machine Learning Process

- Then bring back the original prices.

Predictions	Area m²	Bedroom s	Bathroo ms	Price
\$400,000	180	1	1	\$400,000
\$410,000	180	1	1	\$410,000
\$540,000	210	2	2	\$550,000

Supervised Machine Learning Process

- Finally compare predictions against true test price.

Predictions	Price
\$400,000	
\$410,000	
\$550,000	
\$540,000	

Supervised Machine Learning Process

- This is often labeled as \hat{y} compared again y

\hat{y}	y
Predictions	Price
\$400,000	
\$410,000	
\$550,000	
\$540,000	

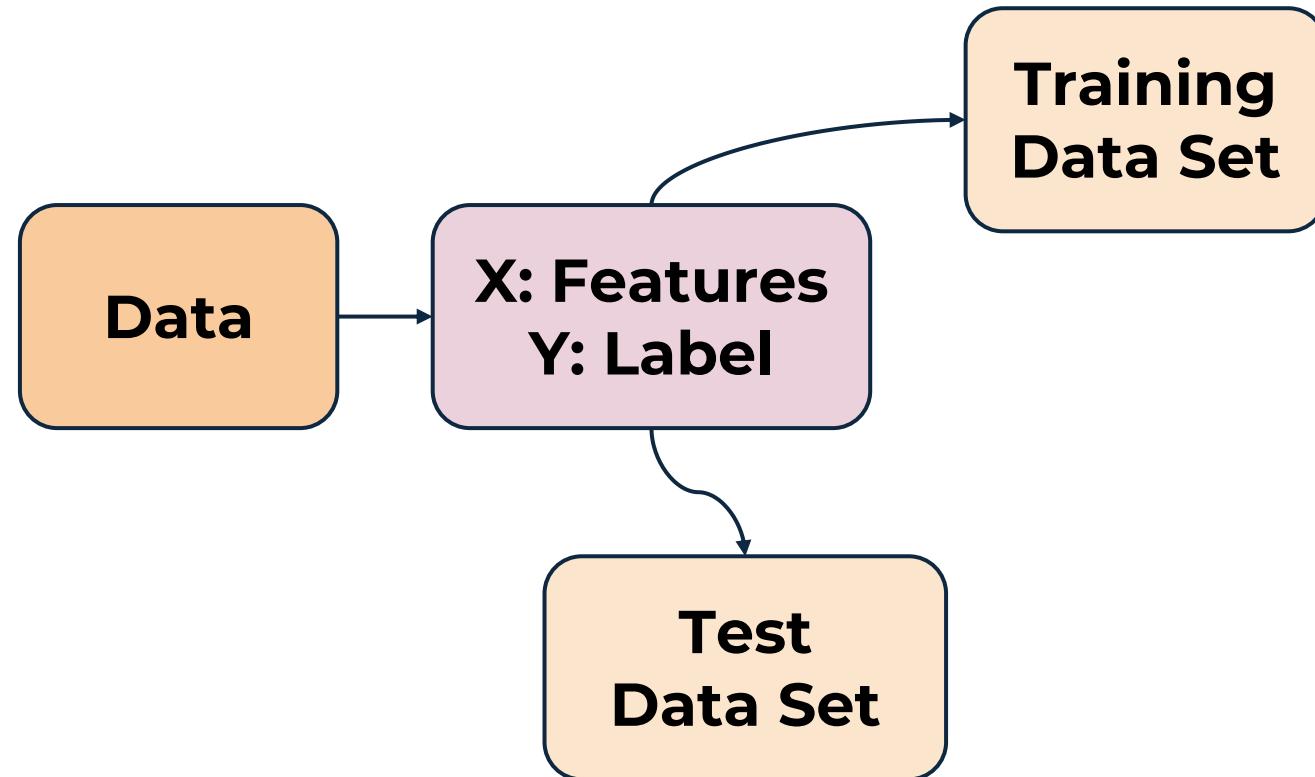
Supervised Machine Learning Process

- Later on we will discuss the many methods of evaluating this performance!

Predictio ns	Price
\$400,000	
\$410,000	
\$540,000	

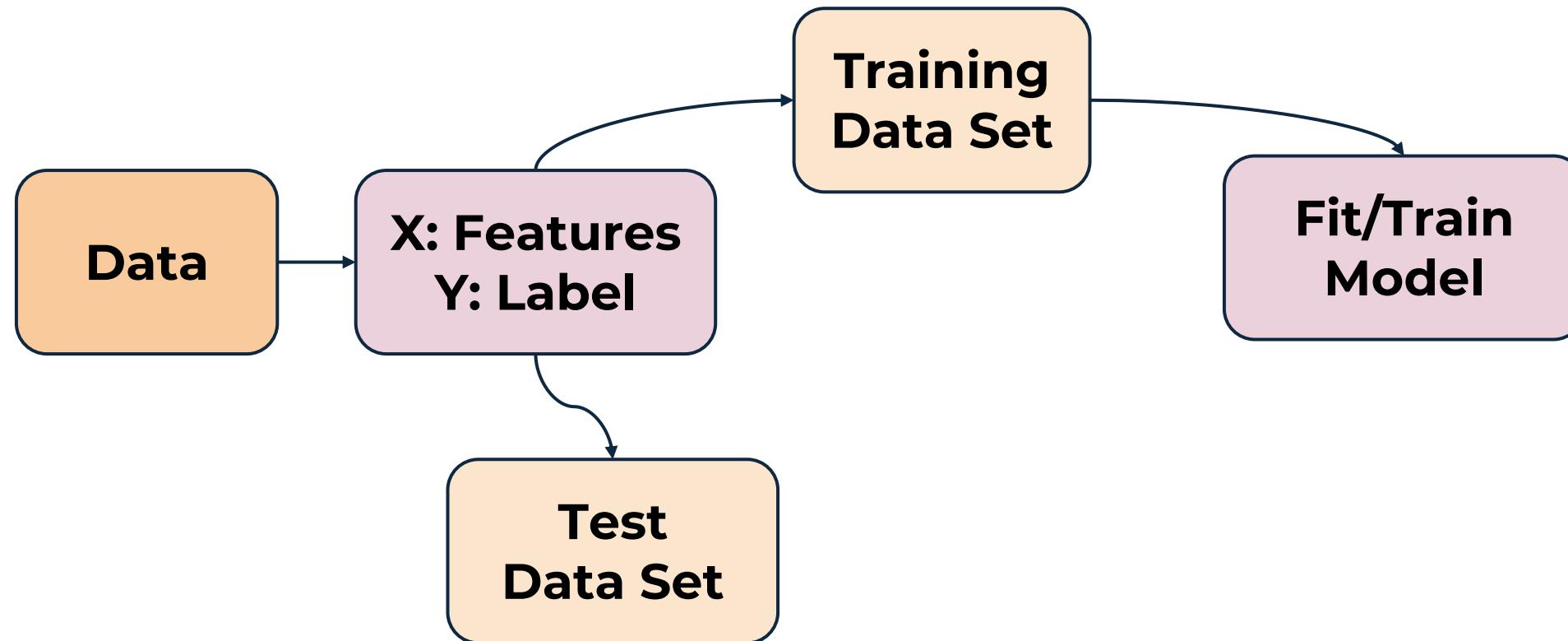
Supervised Machine Learning Process

- Split Data



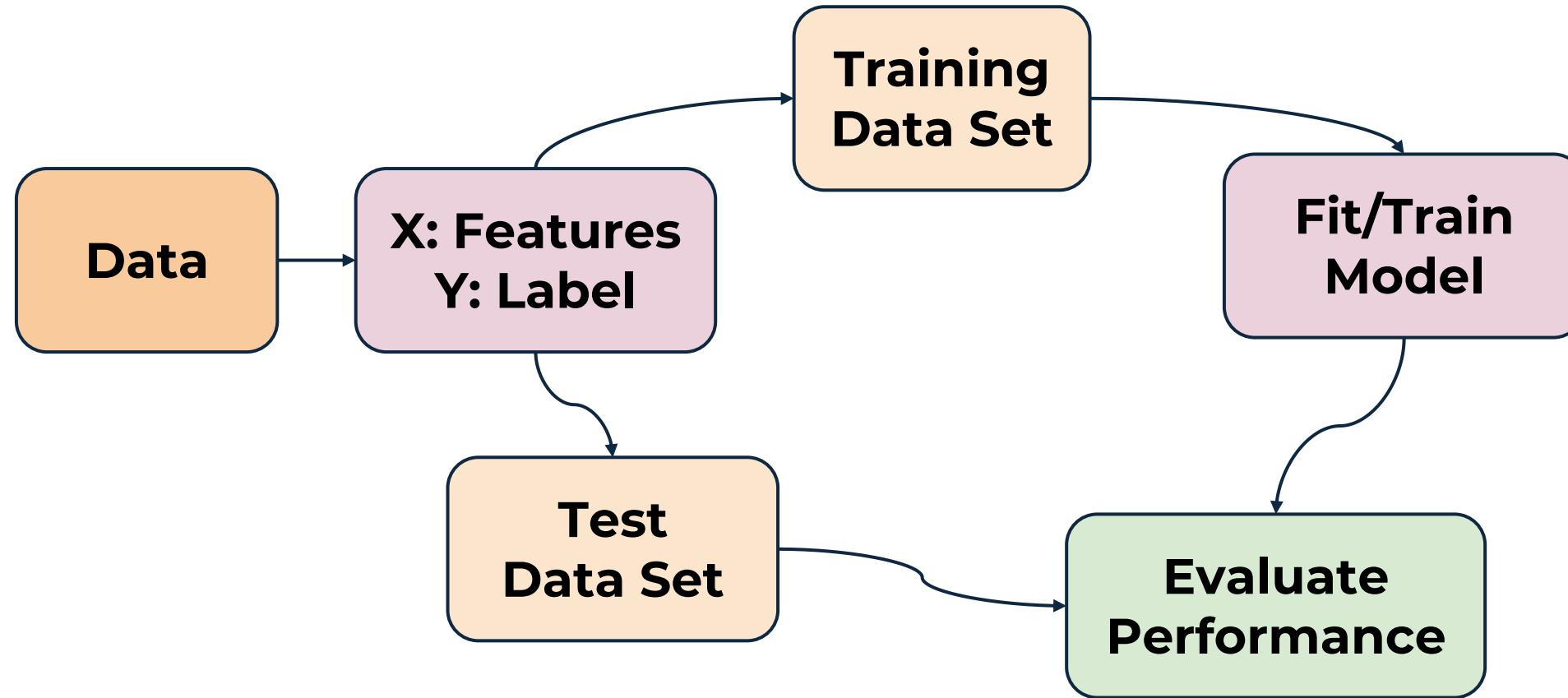
Supervised Machine Learning Process

- Split Data, Fit on Train Data



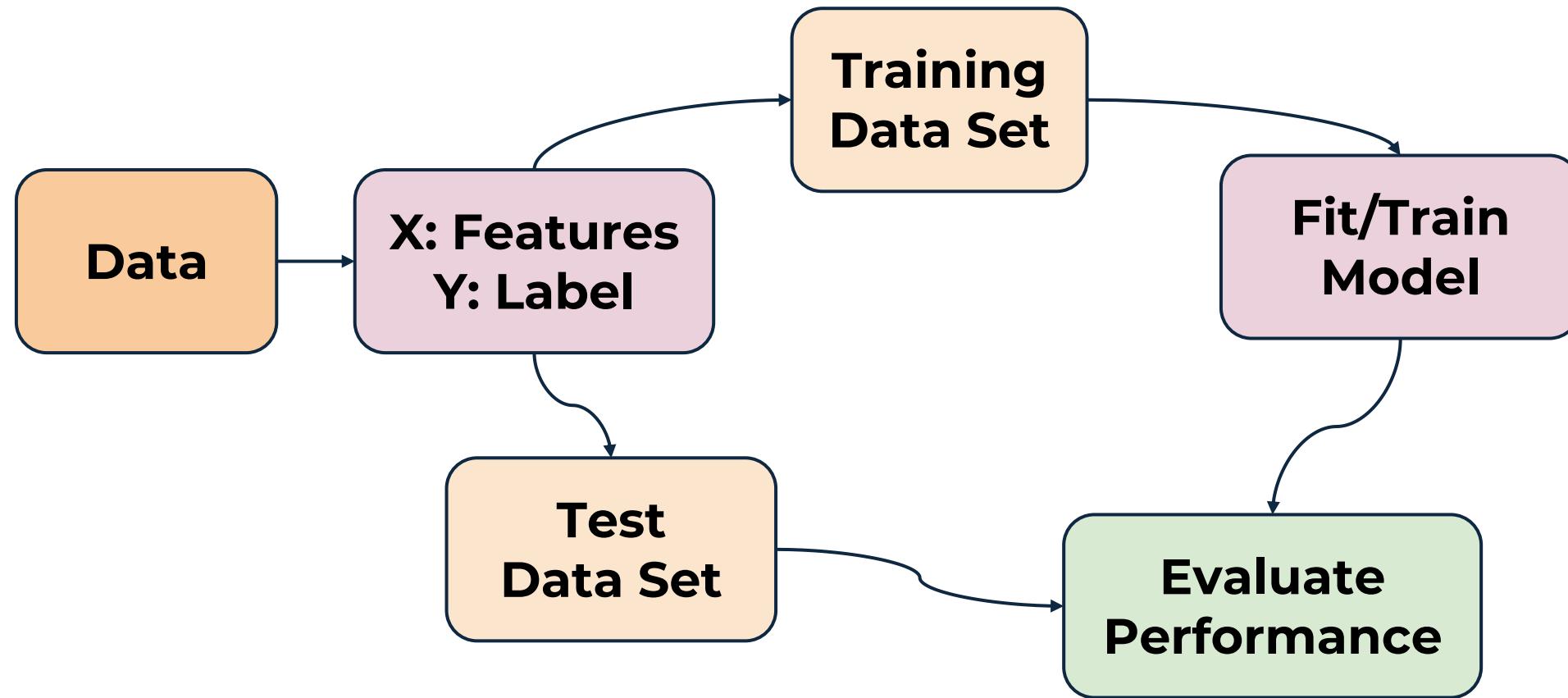
Supervised Machine Learning Process

- Split Data, Fit on Train Data, Evaluate Model



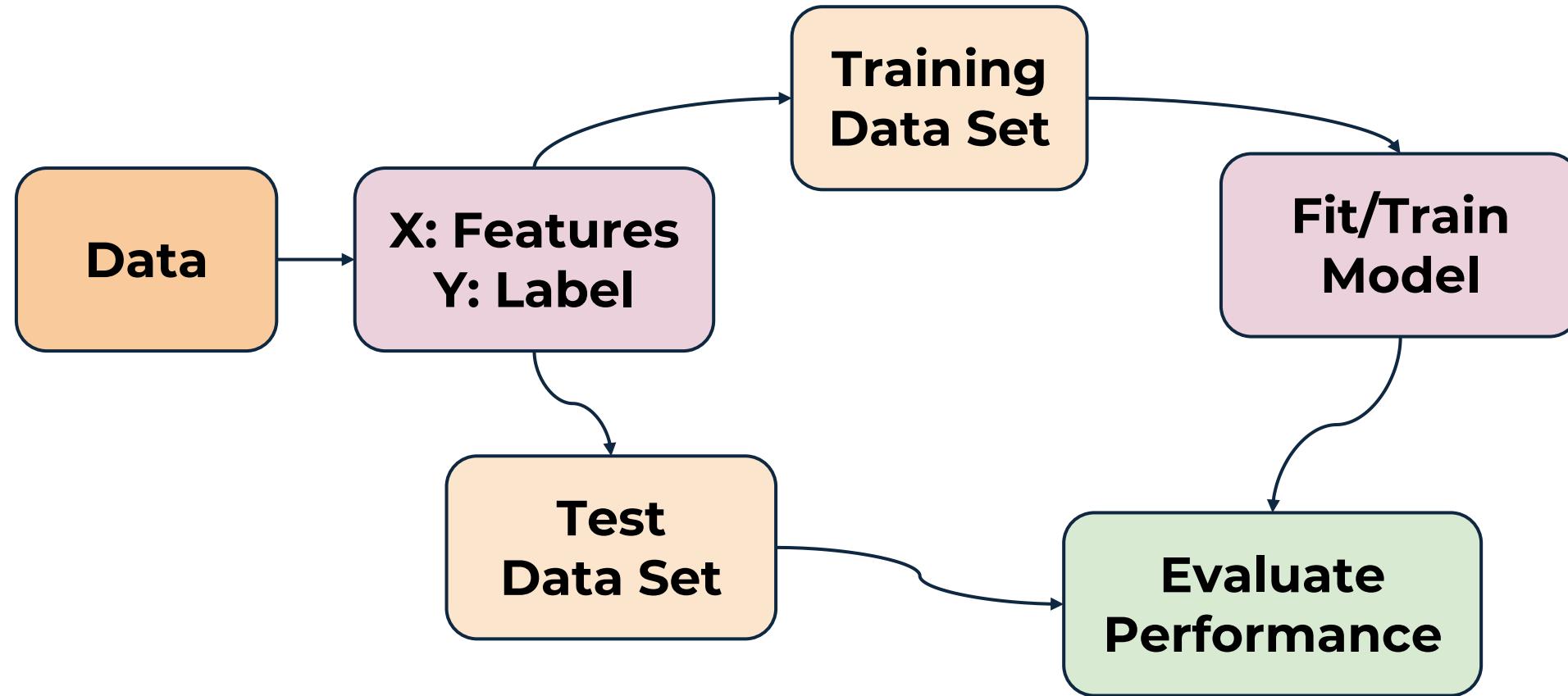
Supervised Machine Learning Process

- What happens if performance isn't great?



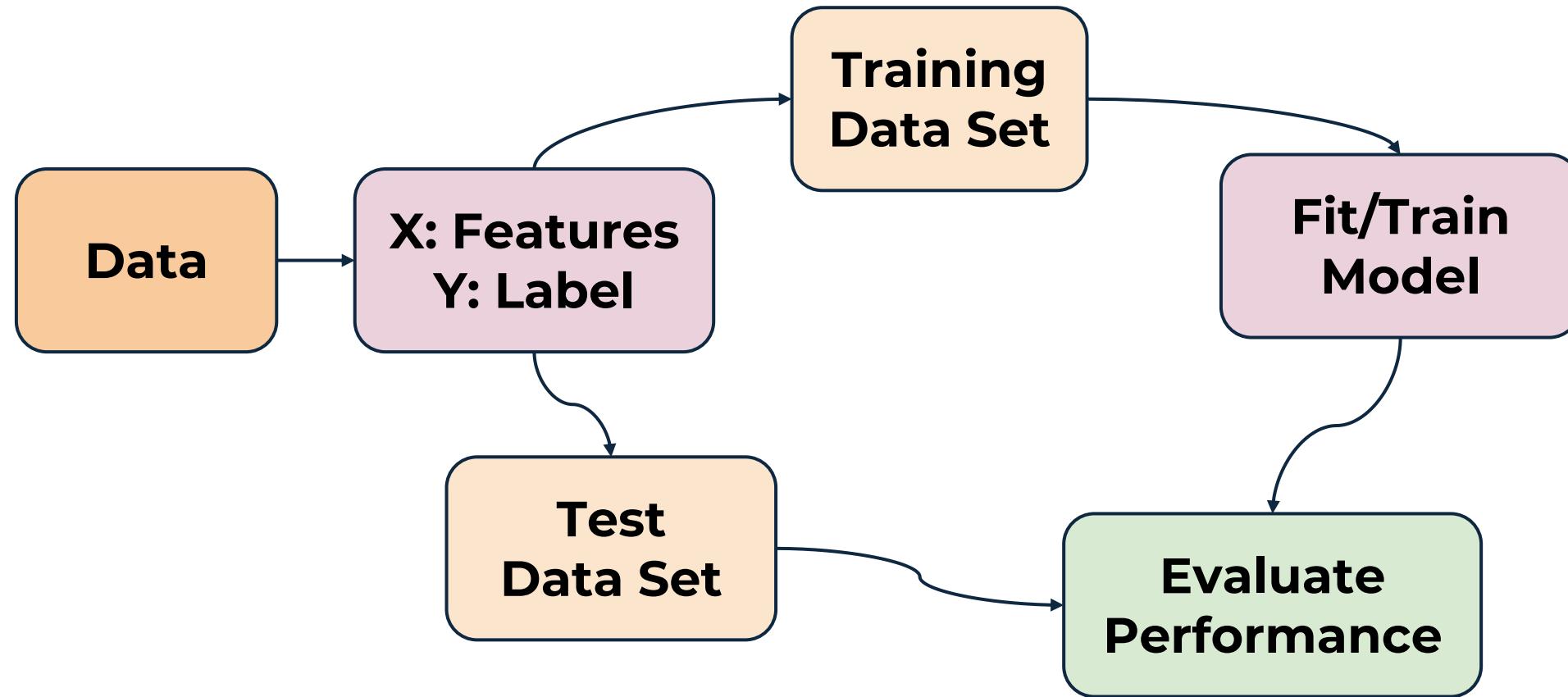
Supervised Machine Learning Process

- We can adjust model **hyperparameters**



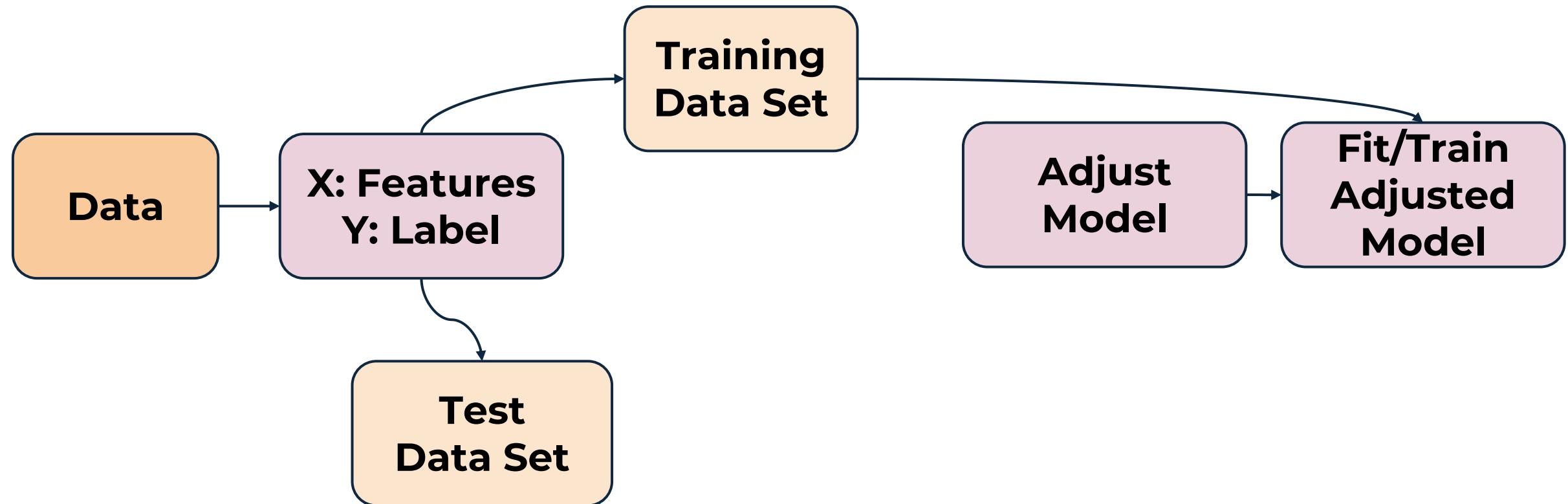
Supervised Machine Learning Process

- Many algorithms have adjustable values



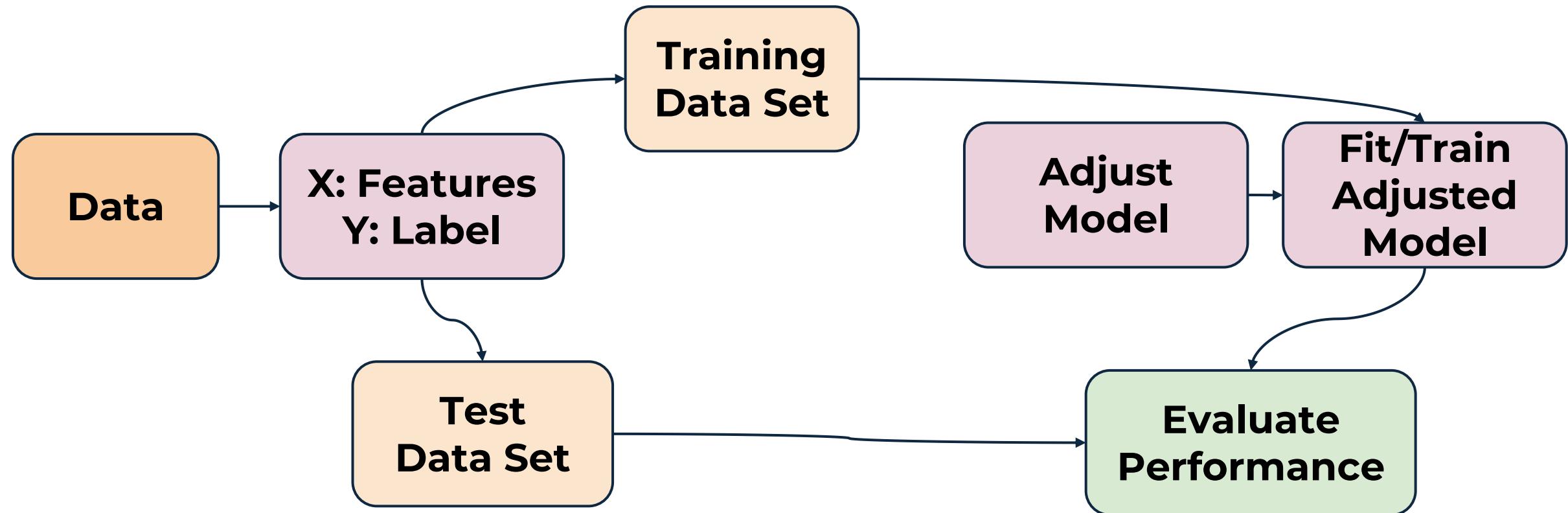
Supervised Machine Learning Process

- Many algorithms have adjustable values



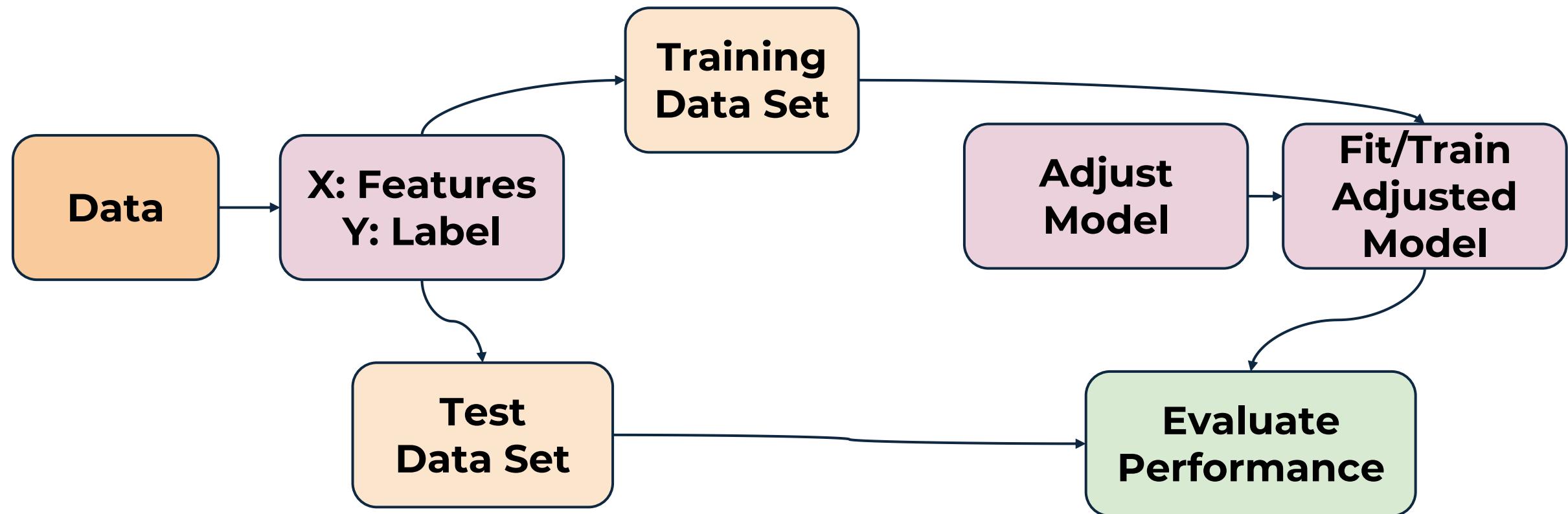
Supervised Machine Learning Process

- Evaluate adjusted model



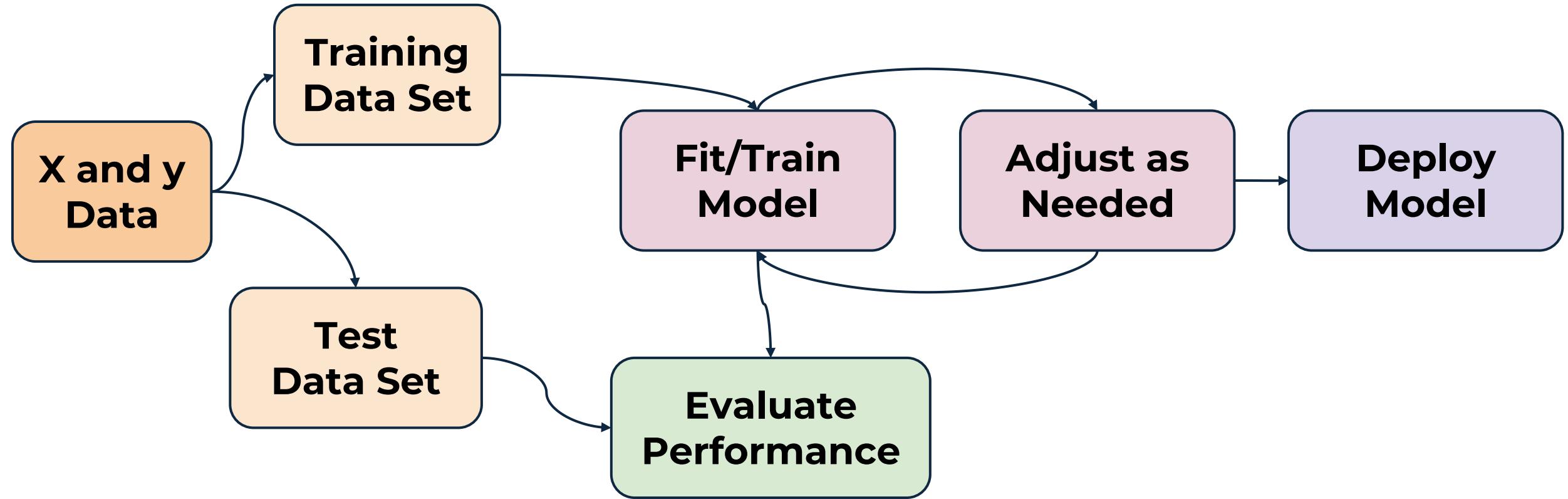
Supervised Machine Learning Process

- Can repeat this process as necessary



Supervised Machine Learning Process

- Full and Simplified Process



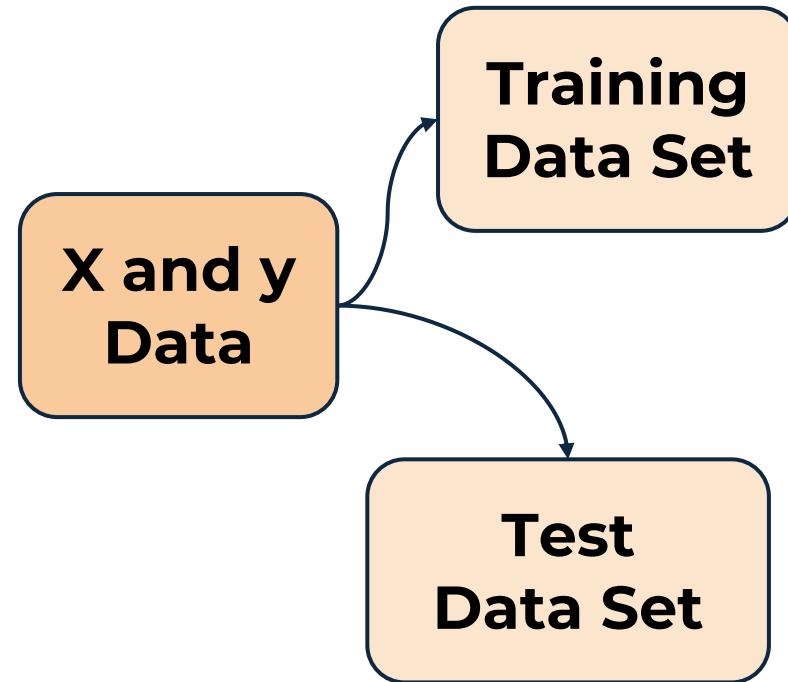
Supervised Machine Learning Process

- Get X and y data

**X and y
Data**

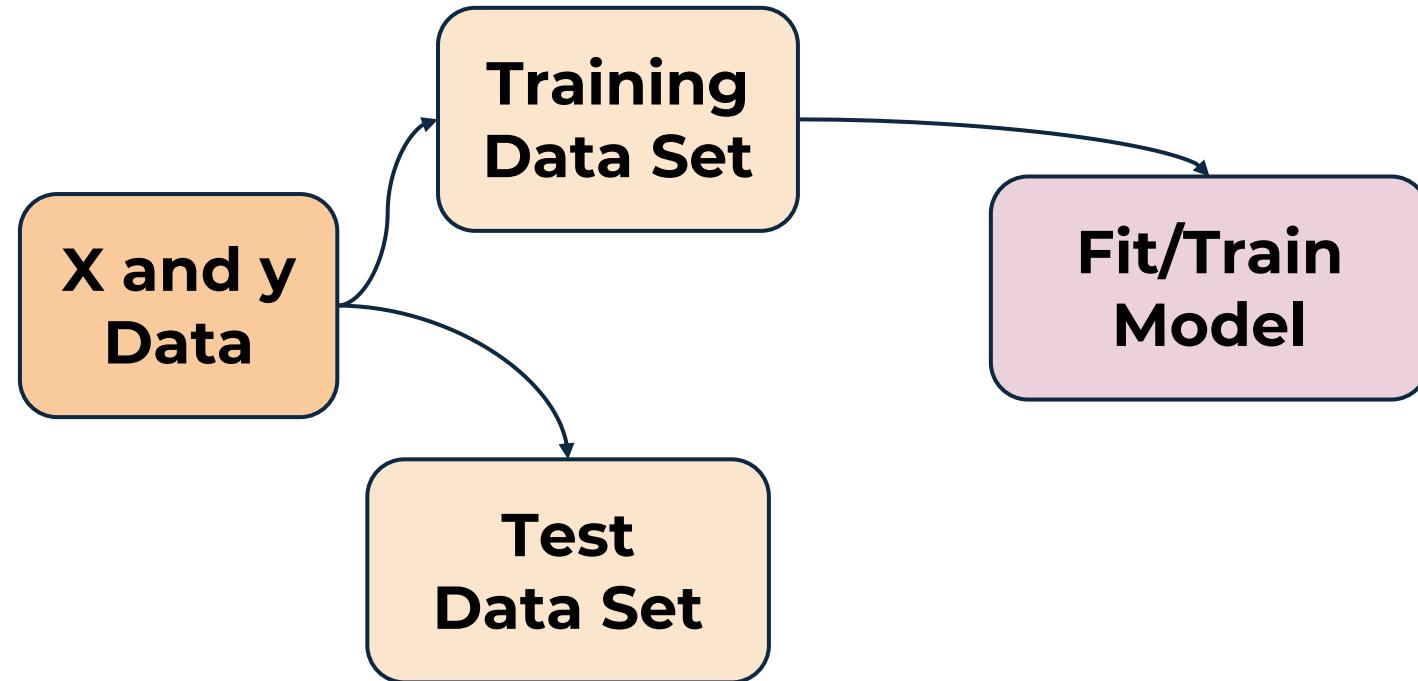
Supervised Machine Learning Process

- Split data for evaluation purposes



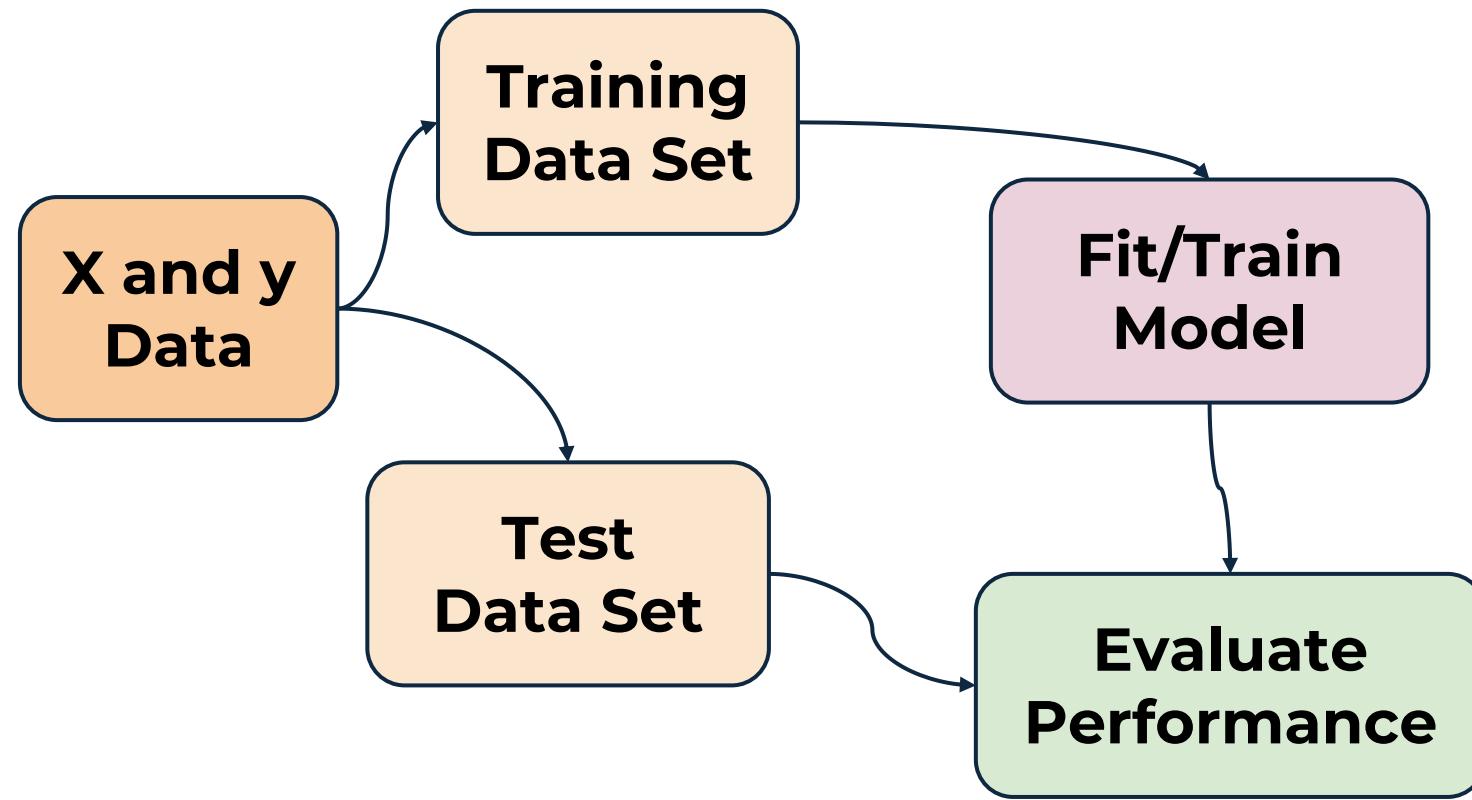
Supervised Machine Learning Process

- Fit ML Model on Training Data Set



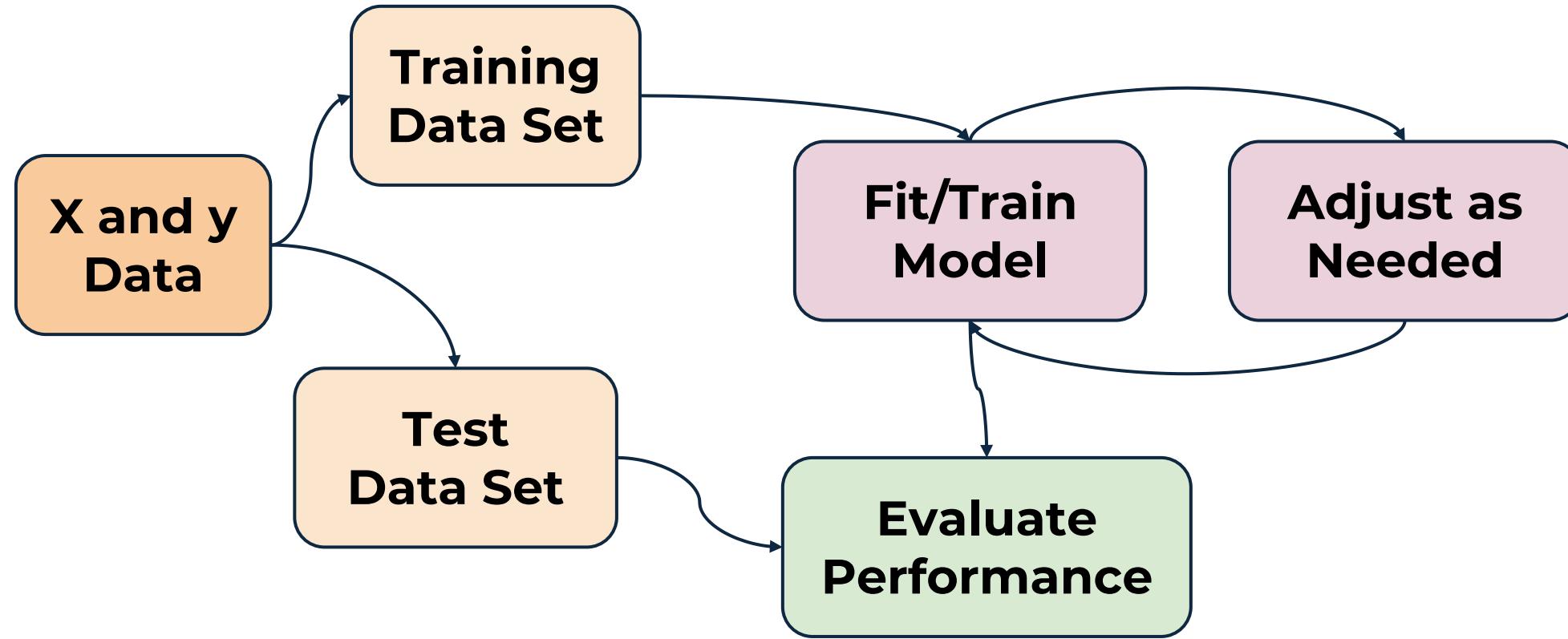
Supervised Machine Learning Process

- Evaluate Model Performance



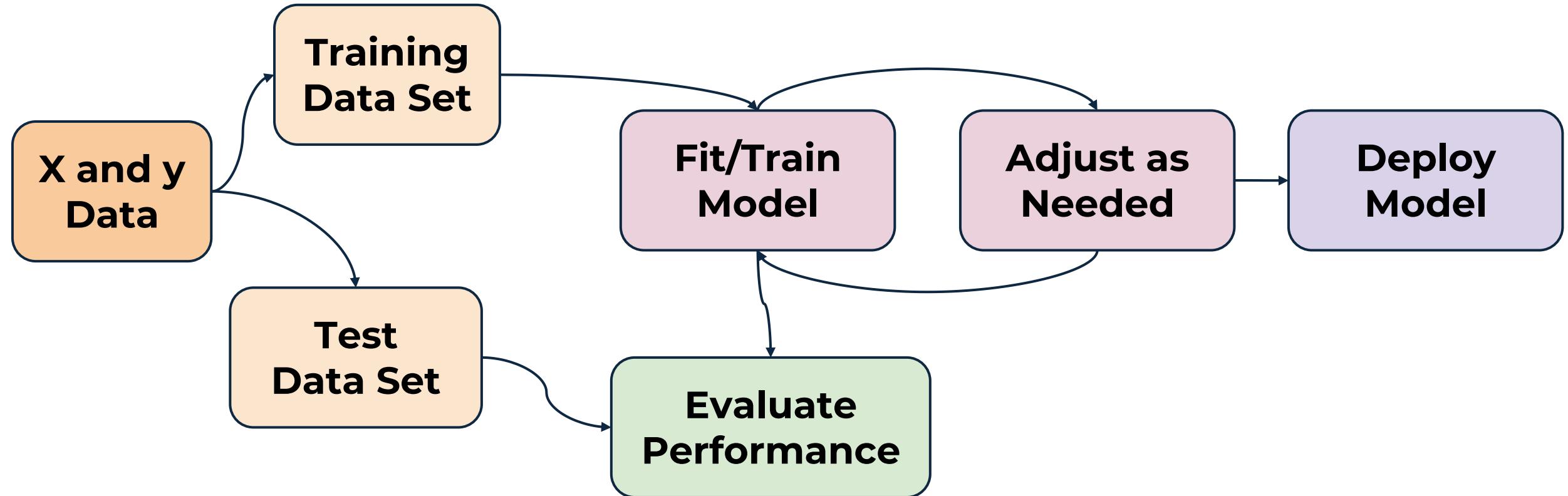
Supervised Machine Learning Process

- Adjust model hyperparameters as needed



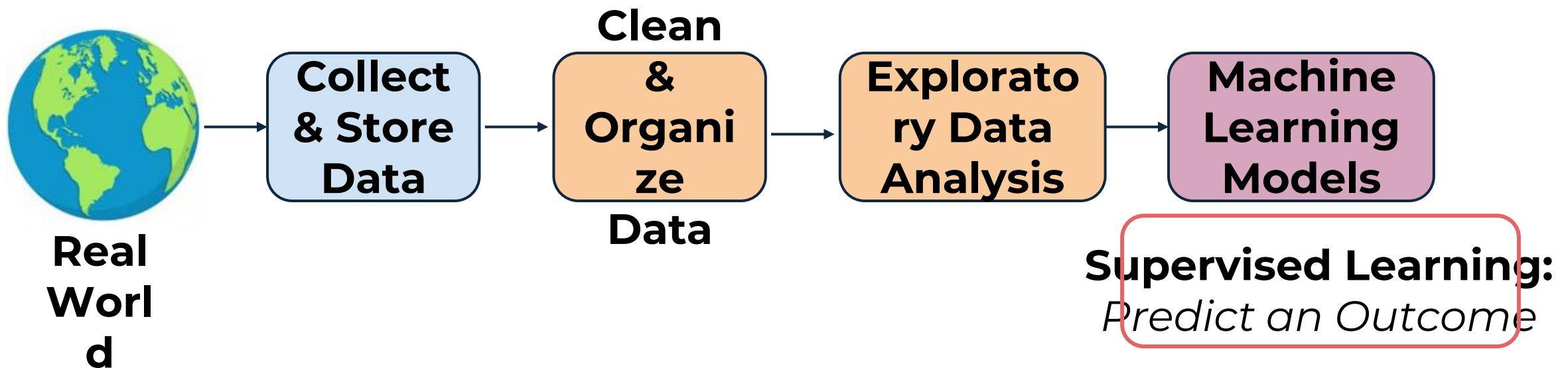
Supervised Machine Learning Process

- Deploy model to real world

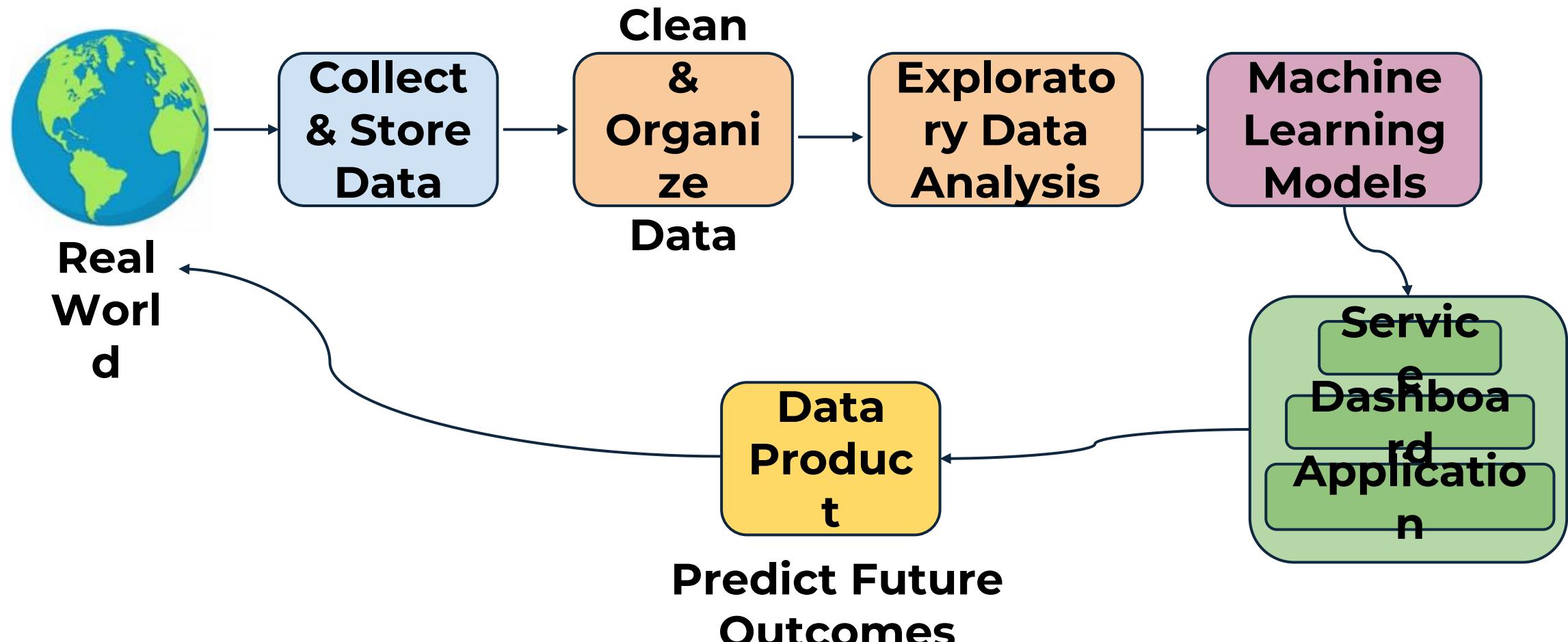


Machine Learning

- ML Process : Supervised Learning Tasks



ML Pathway



Evaluating Performance

CLASSIFICATION

Model Evaluation

- We just learned that after our machine learning process is complete, we will use performance metrics to evaluate how our model did.
- Let's discuss classification metrics in more detail!

Model Evaluation

- The key classification metrics we need to understand are:
 - Accuracy
 - Recall
 - Precision
 - F1-Score

Model Evaluation

- But first, we should understand the reasoning behind these metrics and how they will actually work in the real world!

Model Evaluation

- Typically in any classification task your model can only achieve two results:
 - Either your model was **correct** in its prediction.
 - Or your model was **incorrect** in its prediction.

Model Evaluation

- Fortunately incorrect vs correct expands to situations where you have multiple classes.
- For the purposes of explaining the metrics, let's imagine a **binary classification** situation, where we only have two available classes.

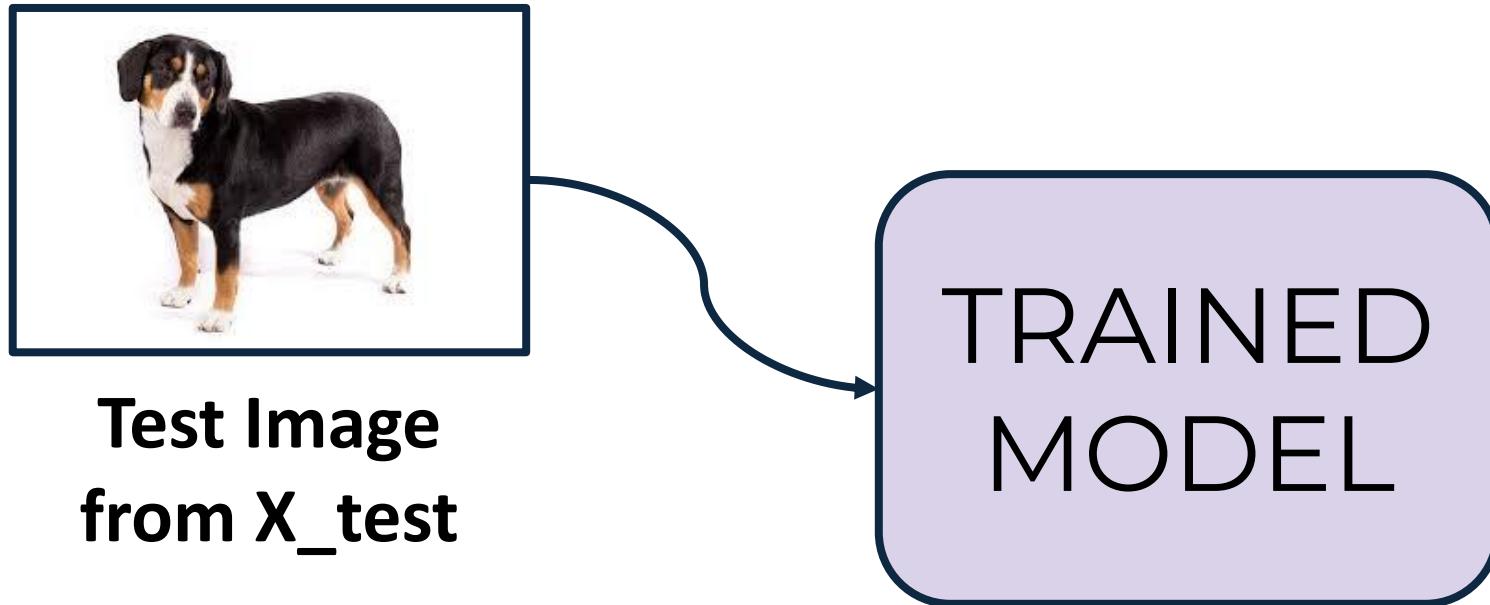
Model Evaluation

- In our example, we will attempt to predict if an image is a dog or a cat.
- Since this is supervised learning, we will first **fit/train** a model on **training data**, then **test** the model on **testing data**.
- Once we have the model's predictions from the **X_test** data, we compare it to the **true y values** (the correct labels).

Model Evaluation

TRAINED
MODEL

Model Evaluation



Model Evaluation



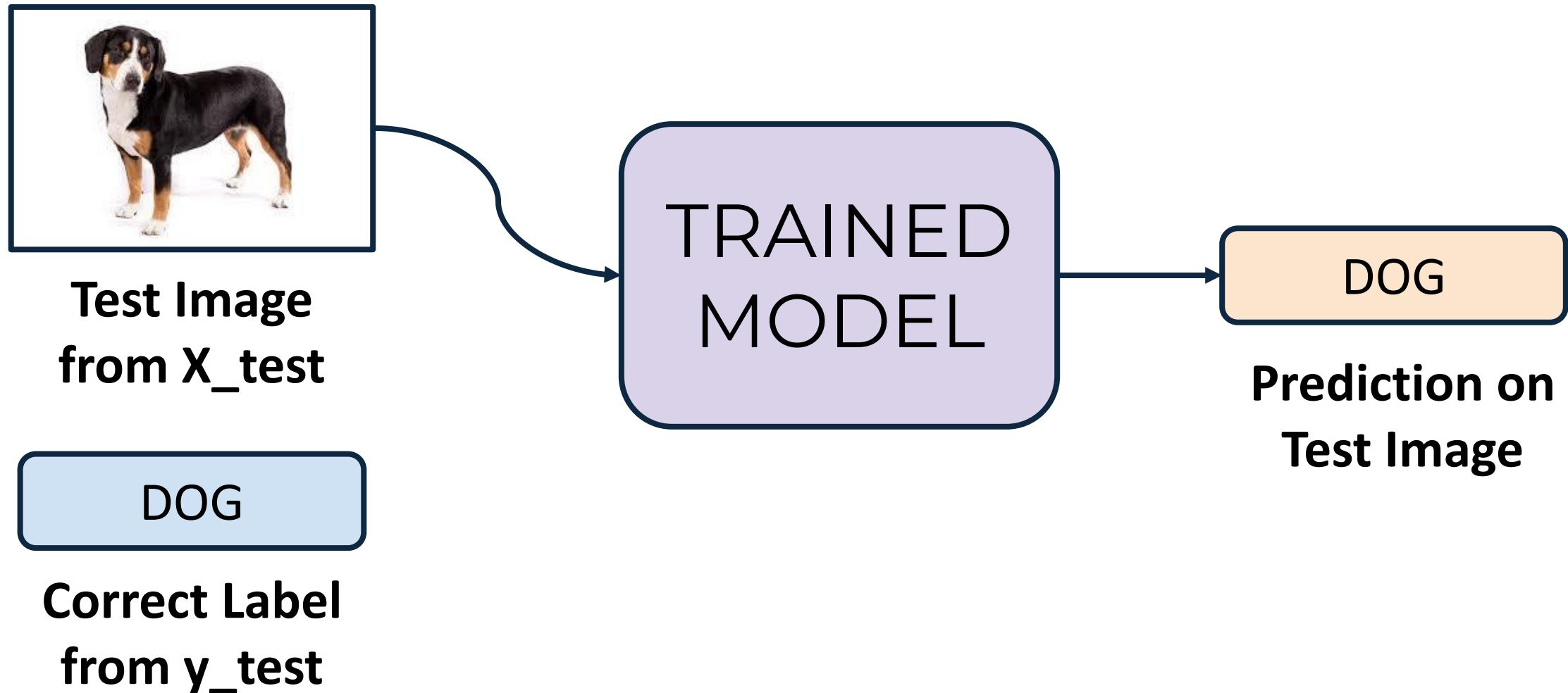
**Test Image
from X_{test}**



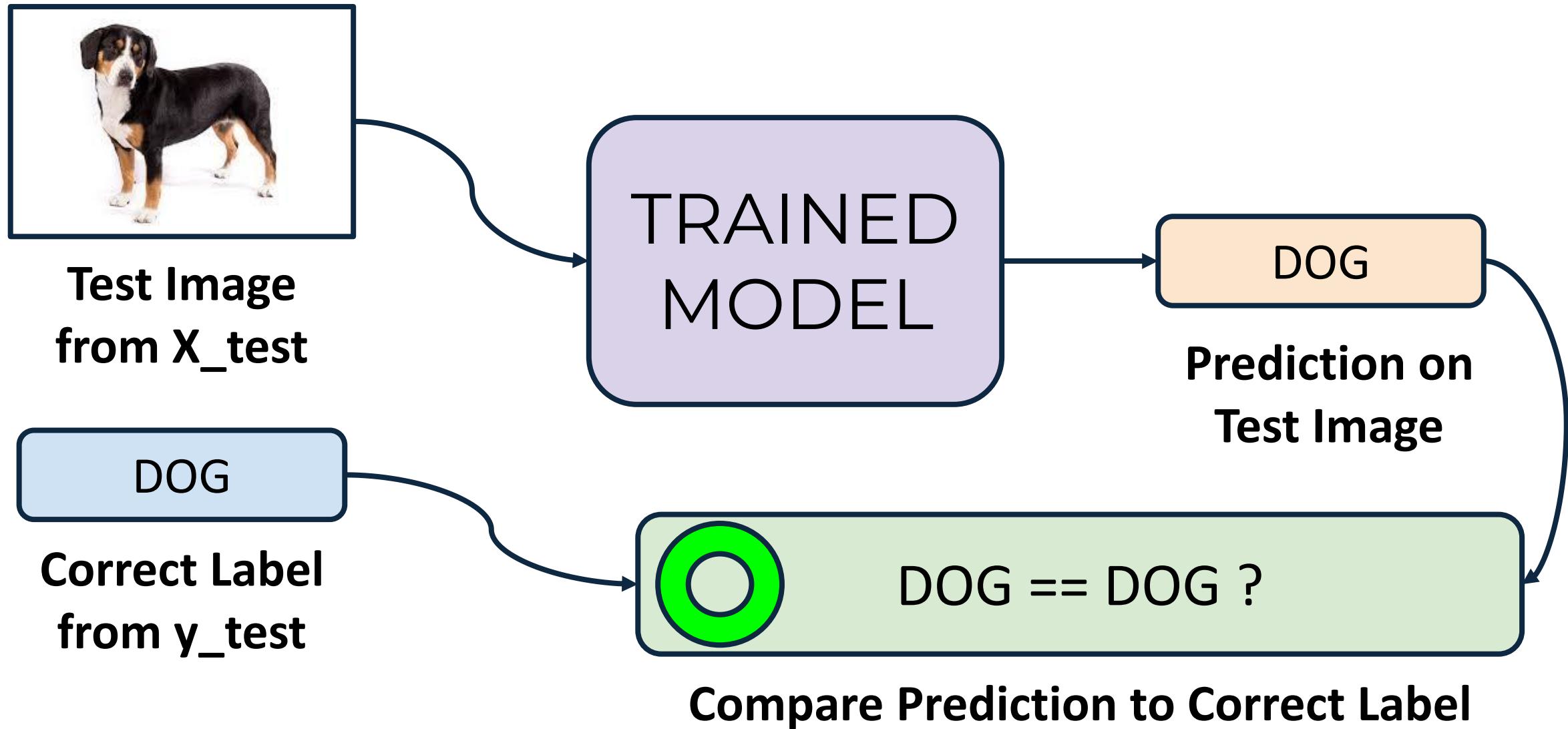
DOG

**Correct Label
from y_{test}**

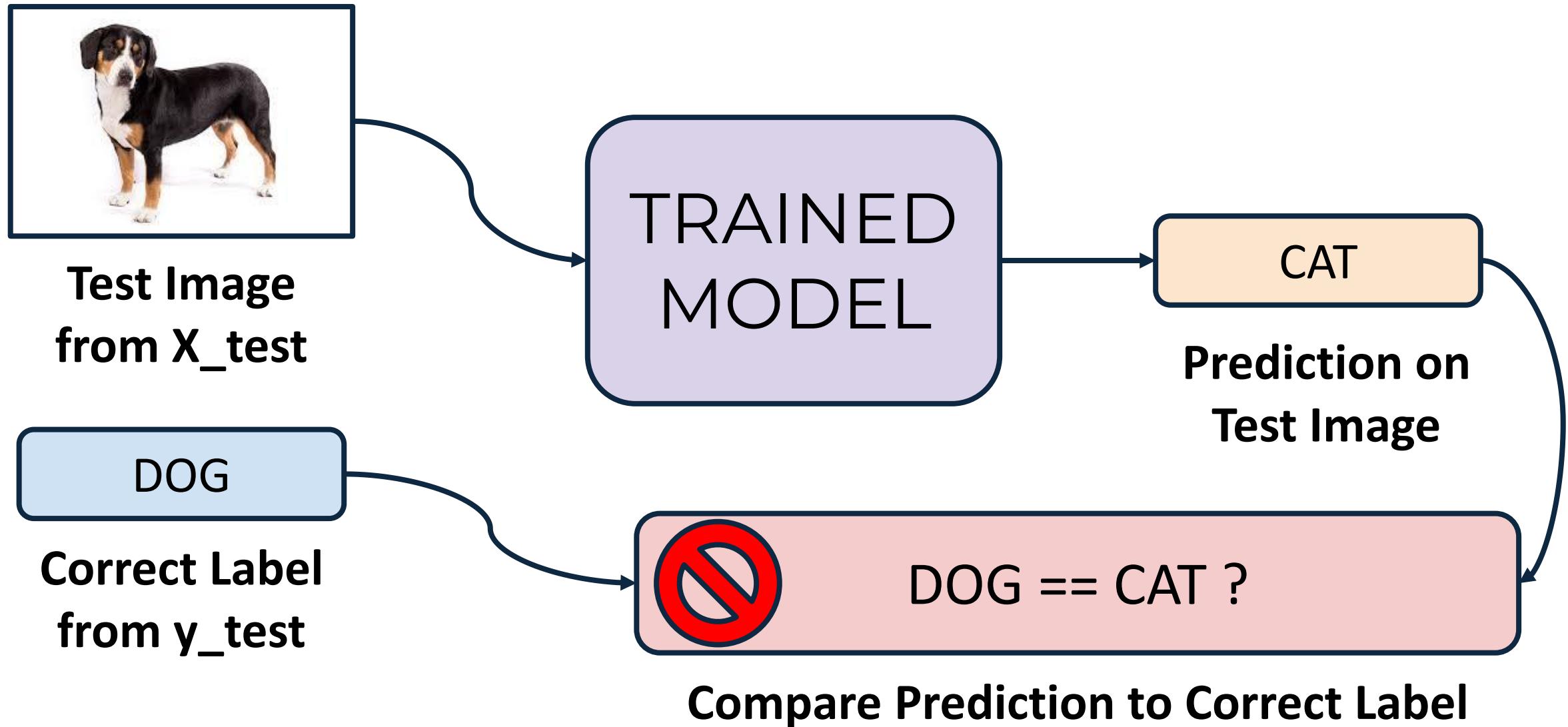
Model Evaluation



Model Evaluation



Model Evaluation



Model Evaluation

- We repeat this process for all the images in our X test data.
- At the end we will have a count of correct matches and a count of incorrect matches.
- The key realization we need to make, is that **in the real world, not all incorrect or correct matches hold equal value!**

Model Evaluation

- Also in the real world, a single metric won't tell the complete story!
- To understand all of this, let's bring back the 4 metrics we mentioned and see how they are calculated.
- We could organize our predicted values compared to the real values in a **confusion matrix**.

Model Evaluation

- Accuracy
 - Accuracy in classification problems is the **number of correct predictions** made by the model divided by the **total number of predictions**.

Model Evaluation

- Accuracy
 - For example, if the X_{test} set was 100 images and our model **correctly** predicted 80 images, then we have **80/100.**
 - **0.8 or 80% accuracy.**

Model Evaluation

- Accuracy
 - Accuracy is useful when target classes are well balanced
 - In our example, we would have roughly the same amount of cat images as we have dog images.

Model Evaluation

- Accuracy
 - Accuracy is **not** a good choice with **unbalanced** classes!
 - Imagine we had 99 images of dogs and 1 image of a cat.
 - If our model was simply a line that always predicted **dog** we would get 99% accuracy!

Model Evaluation

- Accuracy
 - Imagine we had 99 images of dogs and 1 image of a cat.
 - If our model was simply a line that always predicted **dog** we would get 99% accuracy!
 - In this situation we'll want to understand **recall** and **precision**

Model Evaluation

- Recall
 - Ability of a model to find all the relevant cases within a dataset.
 - The precise definition of recall is the number of true positives **divided by** the number of true positives plus the number of false negatives.

Model Evaluation

- Precision
 - Ability of a classification model to identify only the relevant data points.
 - Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives.

Model Evaluation

- Recall and Precision
 - Often you have a trade-off between Recall and Precision.
 - While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points our model says was relevant actually were relevant.

Model Evaluation

- F1-Score
 - In cases where we want to find an optimal blend of precision and recall we can combine the two metrics using what is called the F1 score.

Model Evaluation

- F1-Score
 - The F1 score is the harmonic mean of precision and recall taking both metrics into account in the following equation:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Model Evaluation

- F1-Score
 - We use the harmonic mean instead of a simple average because it punishes extreme values.
 - A classifier with a precision of 1.0 and a recall of 0.0 has a simple average of 0.5 but an F1 score of 0.

Model Evaluation

- We can also view all correctly classified versus incorrectly classified images in the form of a confusion matrix.

Confusion Matrix

		predicted condition	
total population		prediction positive	prediction negative
true condition	condition positive	True Positive (TP)	False Negative (FN) (type II error)
	condition negative	False Positive (FP) (Type I error)	True Negative (TN)

Confusion Matrix

		predicted condition		
		prediction positive	prediction negative	Prevalence $= \frac{\sum \text{condition positive}}{\sum \text{total population}}$
true condition	condition positive	True Positive (TP)	False Negative (FN) (type II error)	True Positive Rate (TPR), Sensitivity, Recall, Probability of Detection $= \frac{\sum \text{TP}}{\sum \text{condition positive}}$
	condition negative	False Positive (FP) (Type I error)	True Negative (TN)	False Positive Rate (FPR), Fall-out, Probability of False Alarm $= \frac{\sum \text{FP}}{\sum \text{condition negative}}$
Accuracy $= \frac{\sum \text{TP} + \sum \text{TN}}{\sum \text{total population}}$	Positive Predictive Value (PPV), Precision $= \frac{\sum \text{TP}}{\sum \text{prediction positive}}$	False Omission Rate (FOR) $= \frac{\sum \text{FN}}{\sum \text{prediction negative}}$	Positive Likelihood Ratio (LR+) $= \frac{\text{TPR}}{\text{FPR}}$	
	False Discovery Rate (FDR) $= \frac{\sum \text{FP}}{\sum \text{prediction positive}}$	Negative Predictive Value (NPV) $= \frac{\sum \text{TN}}{\sum \text{prediction negative}}$	Negative Likelihood Ratio (LR-) $= \frac{\text{FNR}}{\text{TNR}}$	

Model Evaluation

- The main point to remember with the confusion matrix and the various calculated metrics is that they are all fundamentally ways of comparing the predicted values versus the true values.
- What constitutes “good” metrics, will really depend on the specific situation!

Model Evaluation

- Still confused on the confusion matrix?
- No problem! Check out the Wikipedia page for it, it has a really good diagram with all the formulas for all the metrics.
- Throughout the training, we'll usually just print out metrics (e.g. accuracy).

Model Evaluation

- Let's think back on this idea of:
 - What is a good enough accuracy?
- This all depends on the context of the situation!
- Did you create a model to predict presence of a disease?
- Is the disease presence well balanced in the general population? (Probably not!)

Model Evaluation

- Often models are used as quick diagnostic tests to have **before** having a more invasive test (e.g. getting urine test before getting a biopsy)
- We also need to consider what is at stake!

Model Evaluation

- Often we have a precision/recall trade off, We need to decide if the model will should focus on fixing False Positives vs. False Negatives.
- In disease diagnosis, it is probably better to go in the direction of False positives, so we make sure we correctly classify as many cases of disease as possible!

Model Evaluation

- All of this is to say, machine learning is not performed in a “vacuum”, but instead a collaborative process where we should consult with experts in the domain (e.g. medical doctors)

Evaluating Performance

REGRESSION

Evaluating Regression

- Let's take a moment now to discuss evaluating Regression Models
- Regression is a task when a model attempts to predict continuous values (unlike categorical values, which is classification)

Evaluating Regression

- You may have heard of some evaluation metrics like accuracy or recall.
- These sort of metrics aren't useful for regression problems, we need metrics designed for **continuous** values!

Evaluating Regression

- For example, attempting to predict the price of a house given its features is a **regression task**.
- Attempting to predict the country a house is in given its features would be a classification task.

Evaluating Regression

- Let's discuss some of the most common evaluation metrics for regression:
 - Mean Absolute Error
 - Mean Squared Error
 - Root Mean Square Error

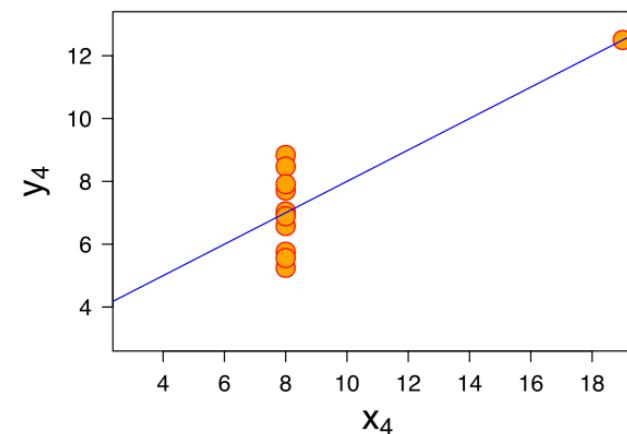
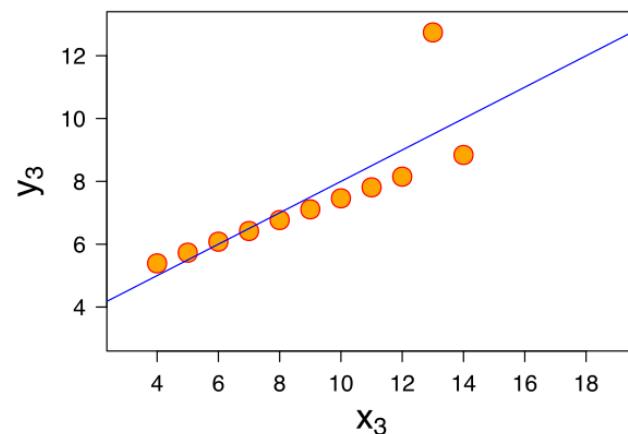
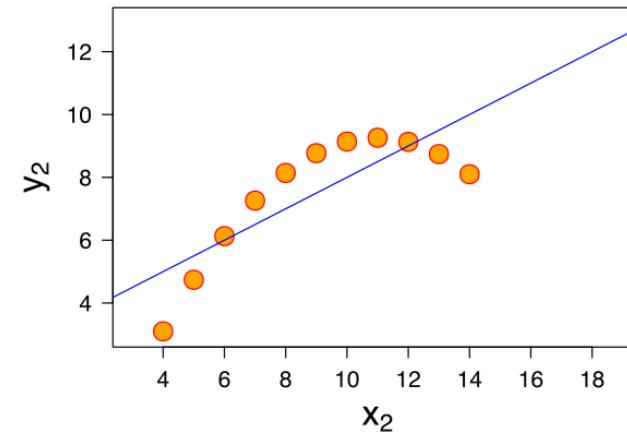
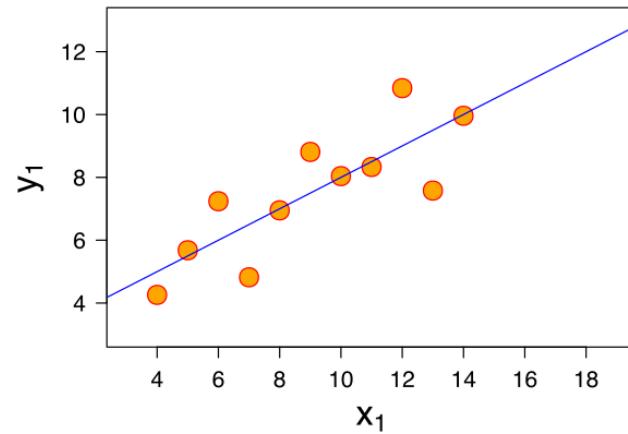
Evaluating Regression

- Mean Absolute Error (MAE)
 - This is the mean of the absolute value of errors.
 - Easy to understand

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

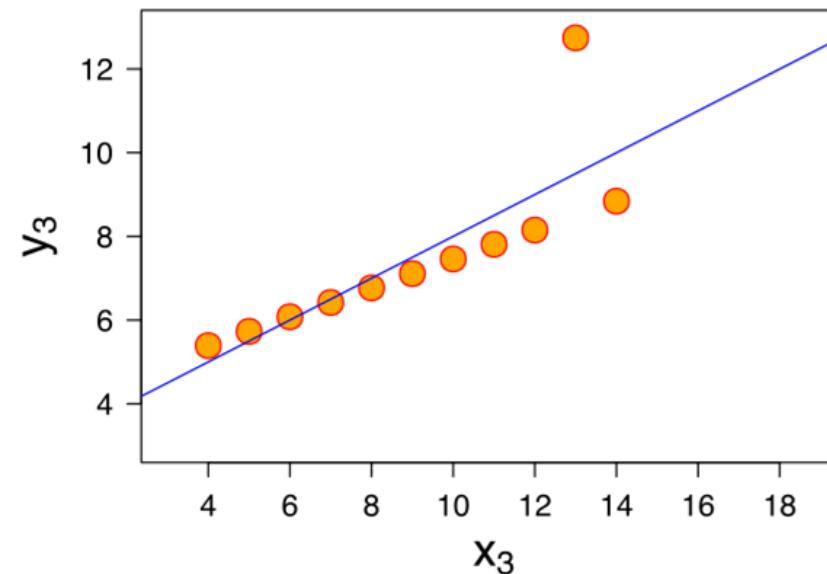
Evaluating Regression

- MAE won't punish large errors however.



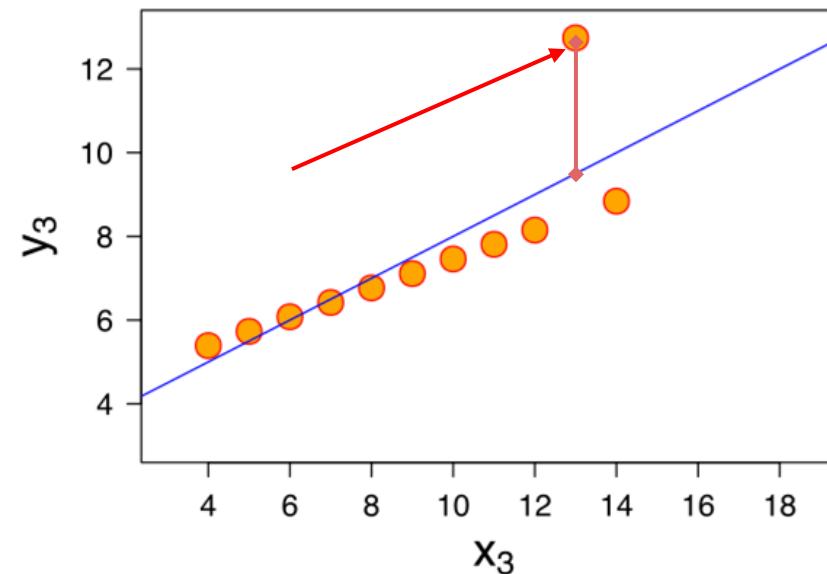
Evaluating Regression

- MAE won't punish large errors however.



Evaluating Regression

- We want our error metrics to account for these!



Evaluating Regression

- Mean Squared Error (MSE)
 - This is the mean of the squared errors.
 - Larger errors are noted more than with MAE, making MSE more popular.

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Evaluating Regression

- Root Mean Square Error (RMSE)
 - This is the root of the mean of the squared errors.
 - Most popular (has same units as y)

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

Machine Learning

- Most common question from students:
 - “Is this value of RMSE good?”
 - Context is everything!
 - A RMSE of \$10 is fantastic for predicting the price of a house, but horrible for predicting the price of a candy bar!

Machine Learning

- Compare your error metric to the average value of the label in your data set to try to get an intuition of its overall performance.
- Domain knowledge also plays an important role here!

Machine Learning

- Context of importance is also necessary to consider.
- We may create a model to predict how much medication to give, in which case small fluctuations in RMSE may actually be very significant.

Machine Learning with Python

Scikit Learn

We will be using the **Scikit Learn** package.

It's the most popular machine learning package for Python and has a lot of algorithms built-in!

Scikit Learn

You'll need to install it using:

conda install scikit-learn

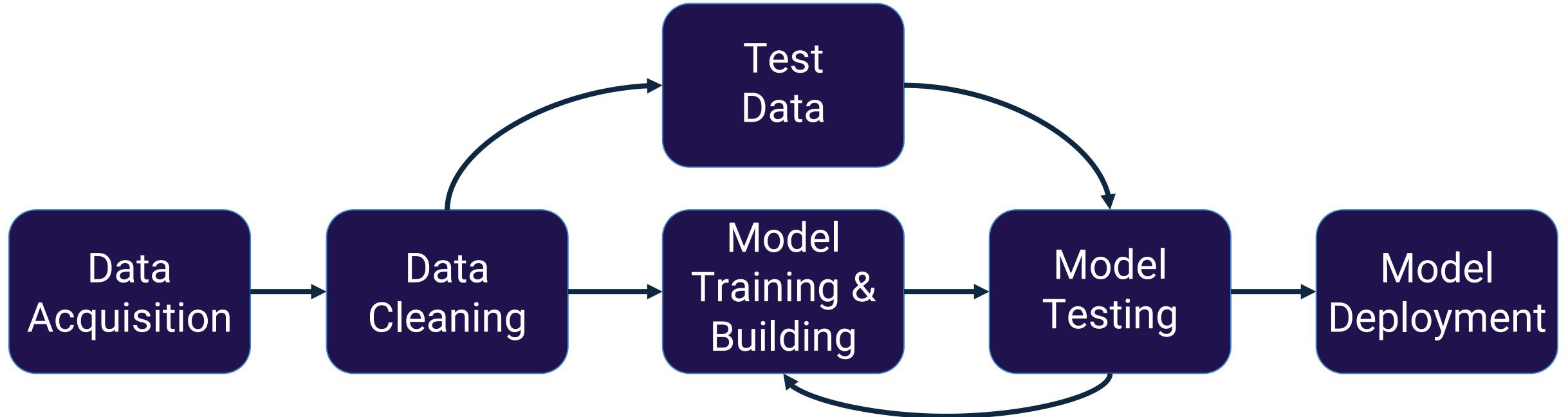
or

pip install scikit-learn

Scikit Learn

- Let's talk about the basic structure of how to use Scikit Learn!
- First, a quick review of the machine learning process.

Machine Learning Process



Scikit Learn

- Now let's go over an example of the process to use SciKit Learn.
- Don't worry about memorizing any of this, we'll get plenty of practice and review when we actually start coding in subsequent lectures!

Scikit Learn

Every algorithm is exposed in scikit-learn via an "Estimator"

First you'll import the model, the general form is:

```
from sklearn.family import Model
```

For example:

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

Scikit Learn

Estimator parameters: All the parameters of an estimator can be set when it is instantiated, and have suitable default values.

You can use Shift+tab in jupyter to check the possible parameters.

Scikit Learn

For example:

```
model = LinearRegression(normalize=True)
```

```
print(model)
```

```
LinearRegression(copy_X=True, fit_intercept=True,  
normalize=True)
```

Scikit Learn

Once you have your model created with your parameters, it is time to fit your model on some data!

But remember, we should split this data into a training set and a test set.

Scikit Learn

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]
```

Scikit Learn

```
>>> x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
>>> x_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> x_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

Scikit Learn

Now that we have split the data, we can train/fit our model on the training data.

This is done through the `model.fit()` method:

```
model.fit(X_train,y_train)
```

Scikit Learn

- Now the model has been fit and trained on the training data.
- The model is ready to predict labels or values on the test set!

Scikit Learn

We get predicted values using the predict method:

```
predictions = model.predict(X_test)
```

Scikit Learn

We can then evaluate our model by comparing our predictions to the correct values.

The evaluation method depends on what sort of machine learning algorithm we are using (e.g. Regression, Classification, Clustering, etc.)

Scikit Learn

Scikit-learn strives to have a uniform interface across all methods, and we'll see examples of these below.

Given a scikit-learn *estimator* object named `model`, the following methods are available...

Scikit Learn

- Available in **all Estimators**
 - `model.fit()` : fit training data.
 - For supervised learning applications, this accepts two arguments: the data X and the labels y (e.g. `model.fit(X, y)`).
 - For unsupervised learning applications, this accepts only a single argument, the data X (e.g. `model.fit(X)`).

Scikit Learn

Available in **supervised estimators**

- `model.predict()` : given a trained model, predict the label of a new set of data. This method accepts one argument, the new data `X_new` (e.g. `model.predict(X_new)`), and returns the learned label for each object in the array.

Scikit Learn

Available in **supervised estimators**

- `model.predict_proba()` : For classification problems, some estimators also provide this method, which returns the probability that a new observation has each categorical label. In this case, the label with the highest probability is returned by `model.predict()`.

Scikit Learn

Available in **supervised estimators**

- `model.score()` : for classification or regression problems, most estimators implement a `score` method. Scores are between 0 and 1, with a larger score indicating a better fit.

Scikit Learn

Available in **unsupervised estimators**

- `model.predict()` : predict labels in clustering algorithms.

Scikit Learn

Available in **unsupervised estimators**

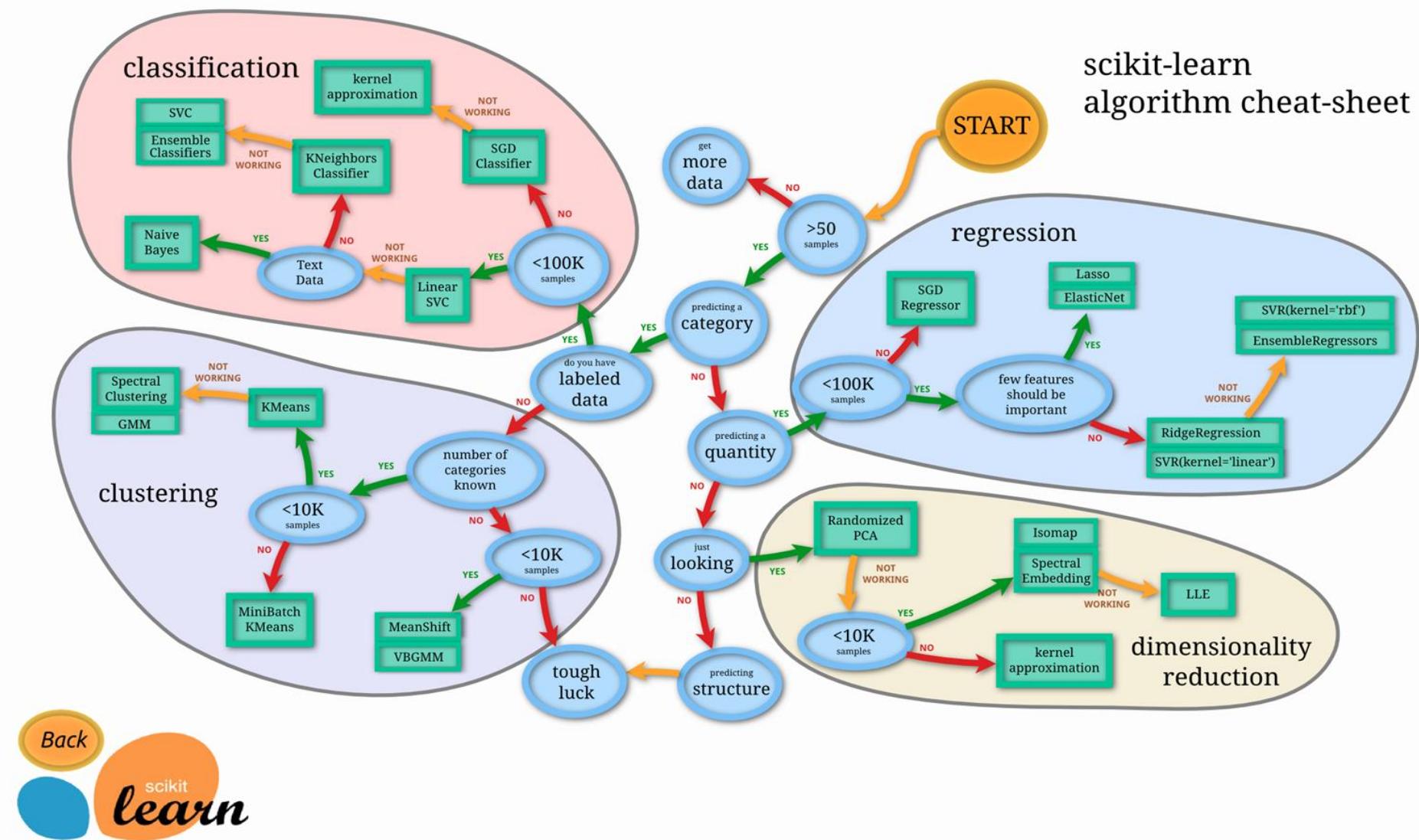
- `model.transform()` : given an unsupervised model, transform new data into the new basis. This also accepts one argument `X_new`, and returns the new representation of the data based on the unsupervised model.

Scikit Learn

Available in **unsupervised estimators**

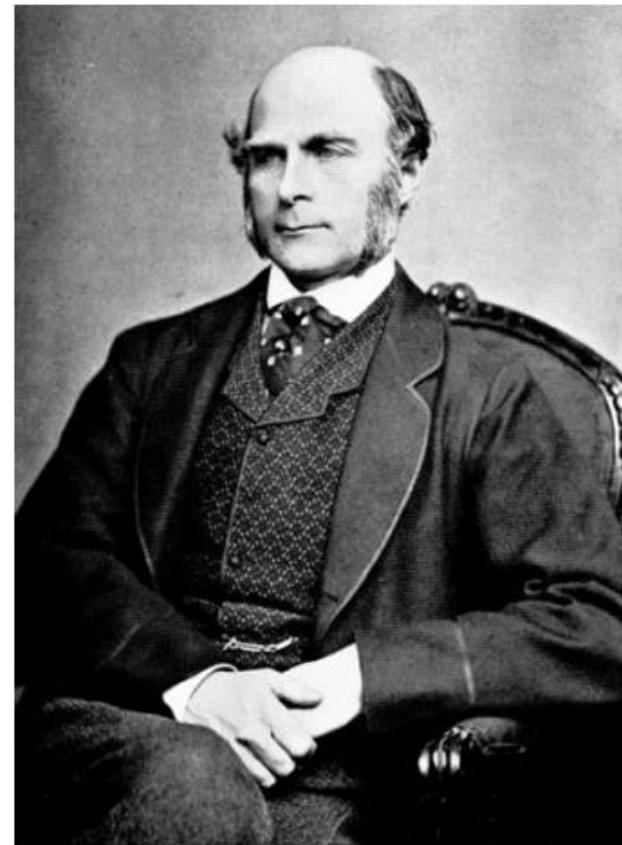
- `model.fit_transform()` : some estimators implement this method, which more efficiently performs a fit and a transform on the same input data.

Choosing an Algorithm



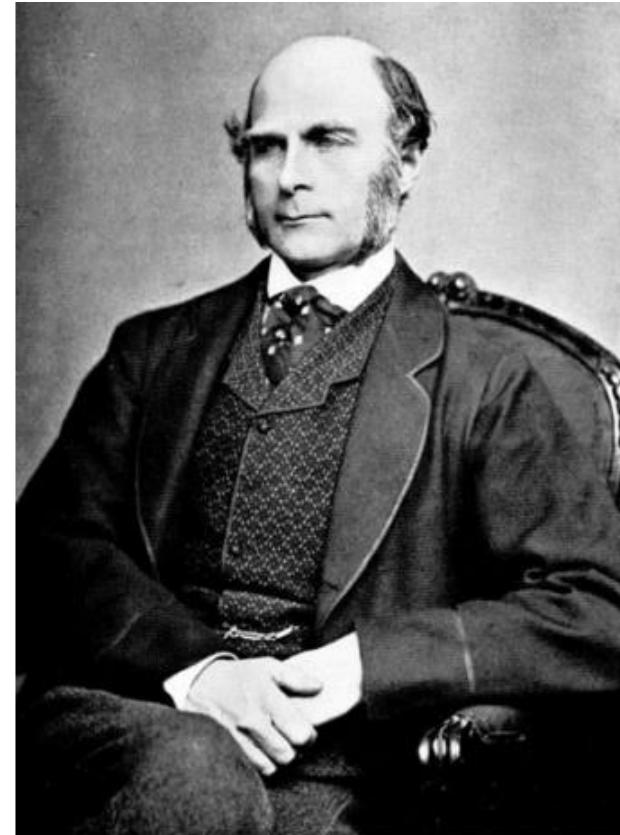
Introduction to Linear Regression

This all started in the 1800s with a guy named [Francis Galton](#). Galton was studying the relationship between parents and their children. In particular, he investigated the relationship between the heights of fathers and their sons.



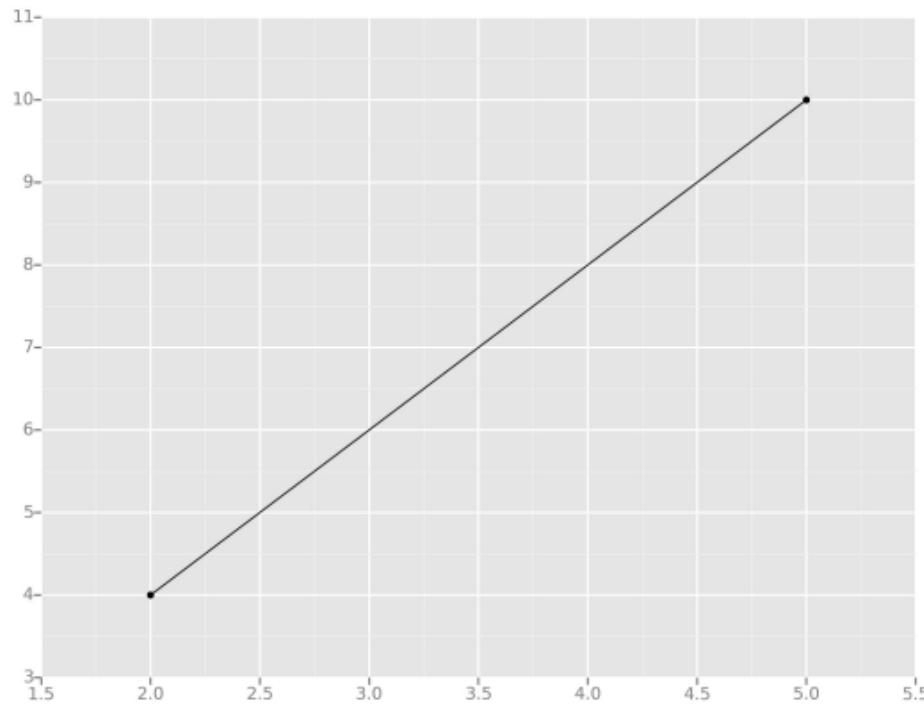
What he discovered was that a man's son tended to be roughly as tall as his father.

However Galton's breakthrough was that the son's height **tended to be closer to the overall average** height of all people.



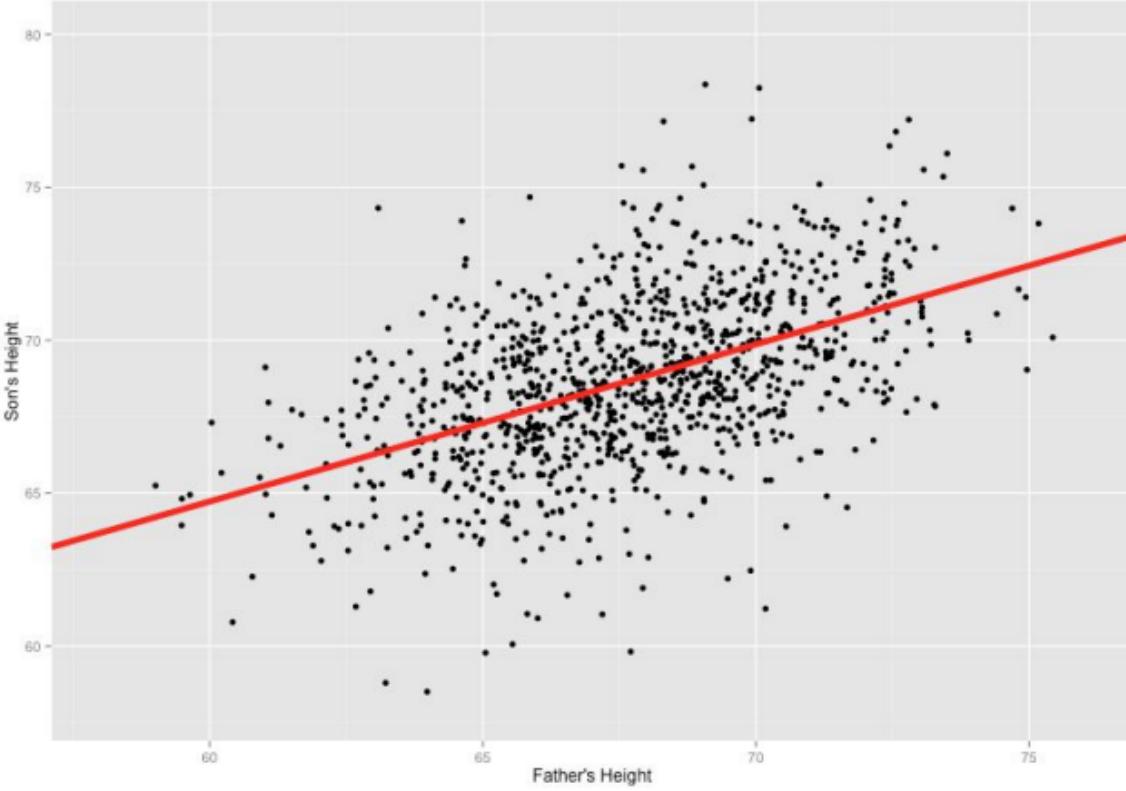
All we're trying to do when we calculate our regression line is draw a line that's as close to every dot as possible.

For classic linear regression, or "Least Squares Method", you only measure the closeness in the "up and down" direction

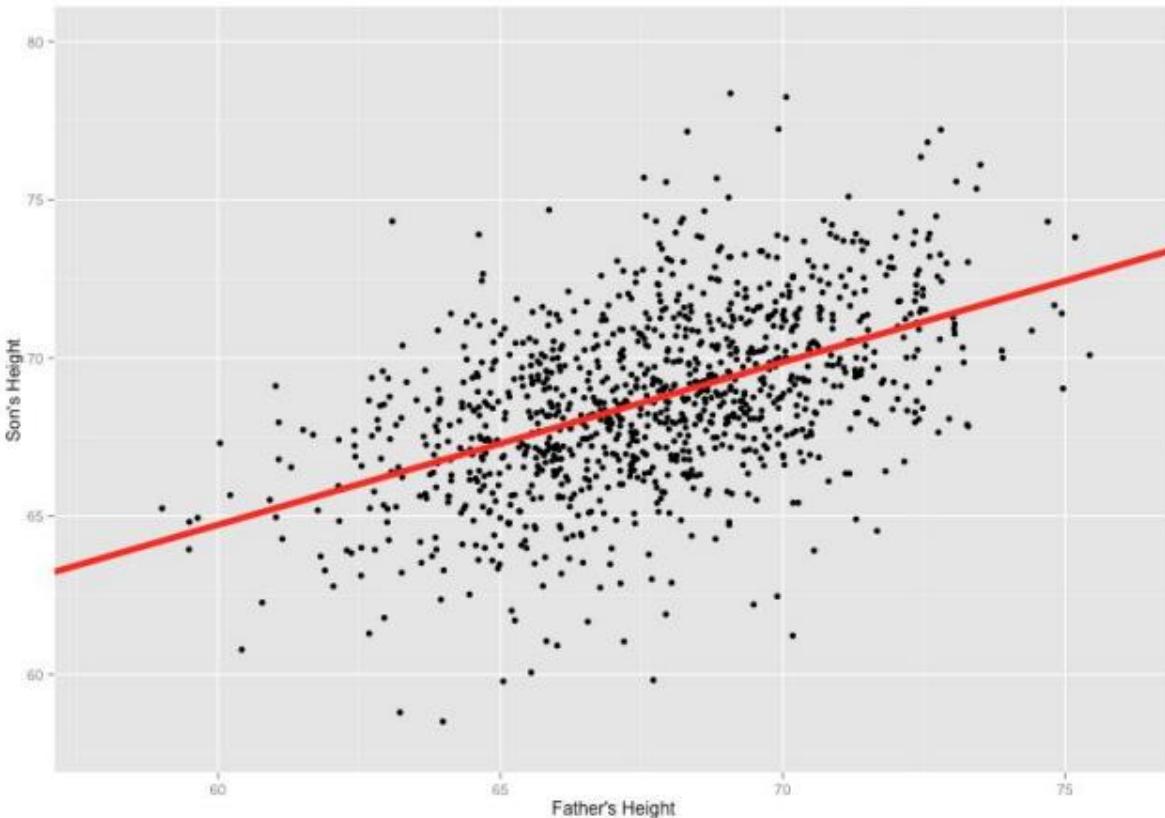


Our goal with linear regression is to **minimize the vertical distance** between all the data points and our line.

So in determining the **best line**, we are attempting to minimize the distance between **all** the points and their distance to our line.

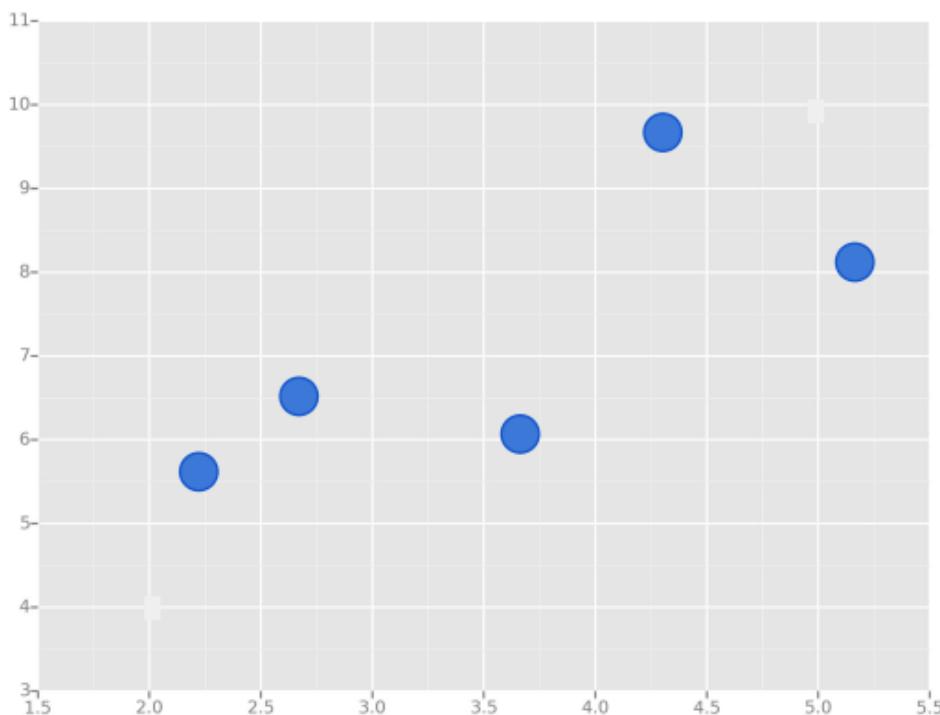


There are lots of different ways to minimize this, (sum of squared errors, sum of absolute errors, etc), but all these methods have a general goal of minimizing this distance.



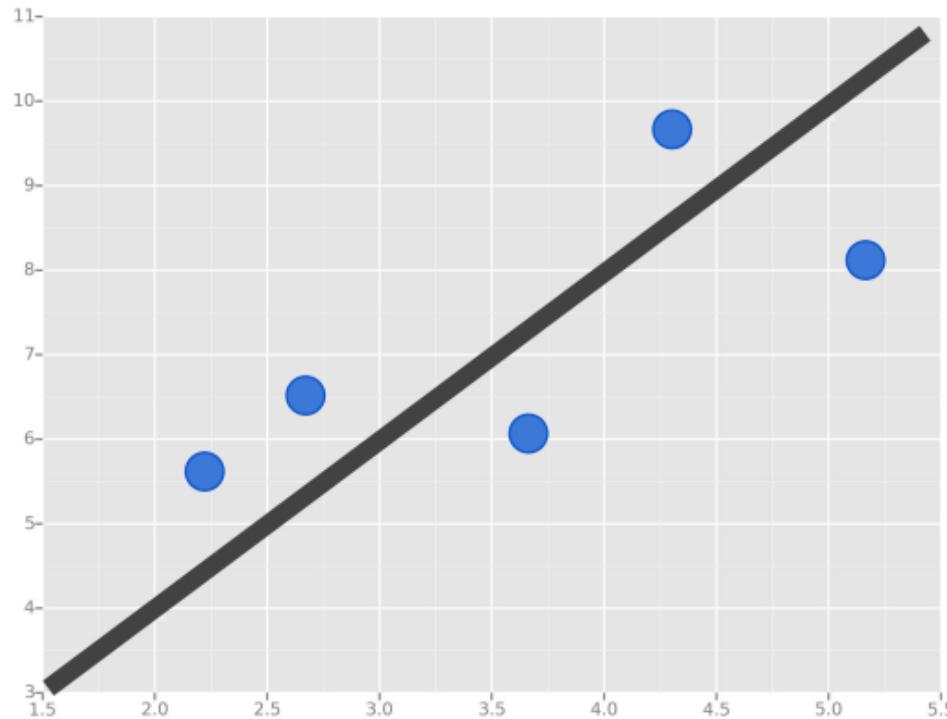
For example, one of the most popular methods is the least squares method.

Here we have blue data points along an x and y axis.



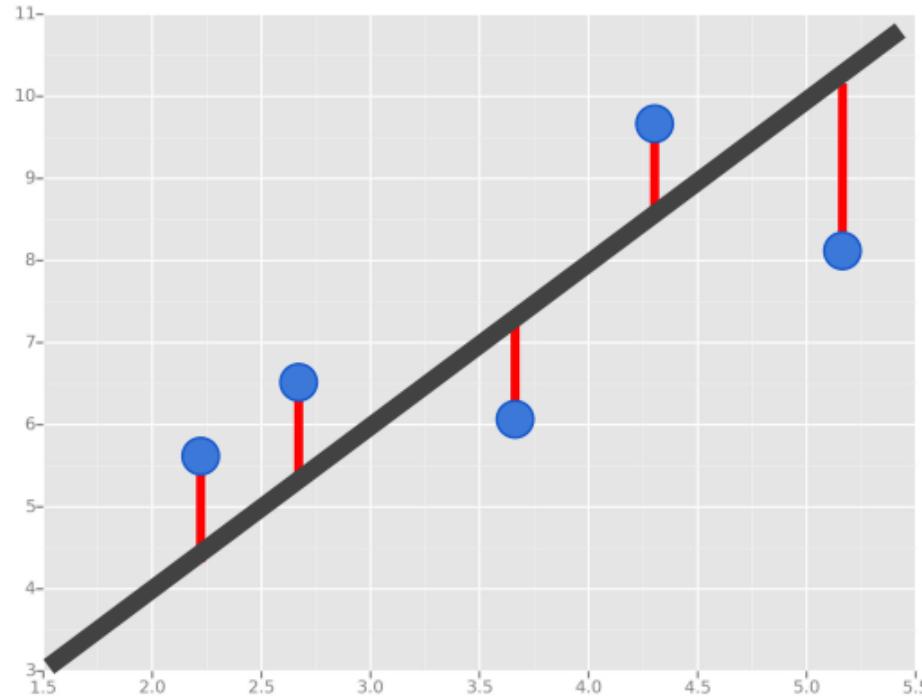
Now we want to fit a linear regression line.

The question is, how do we decide which line is the best fitting one?



We'll use the Least Squares Method, which is fitted by minimizing the ***sum of squares of the residuals***.

The residuals for an observation is the difference between the observation (the y-value) and the fitted line.



Polynomial Regression

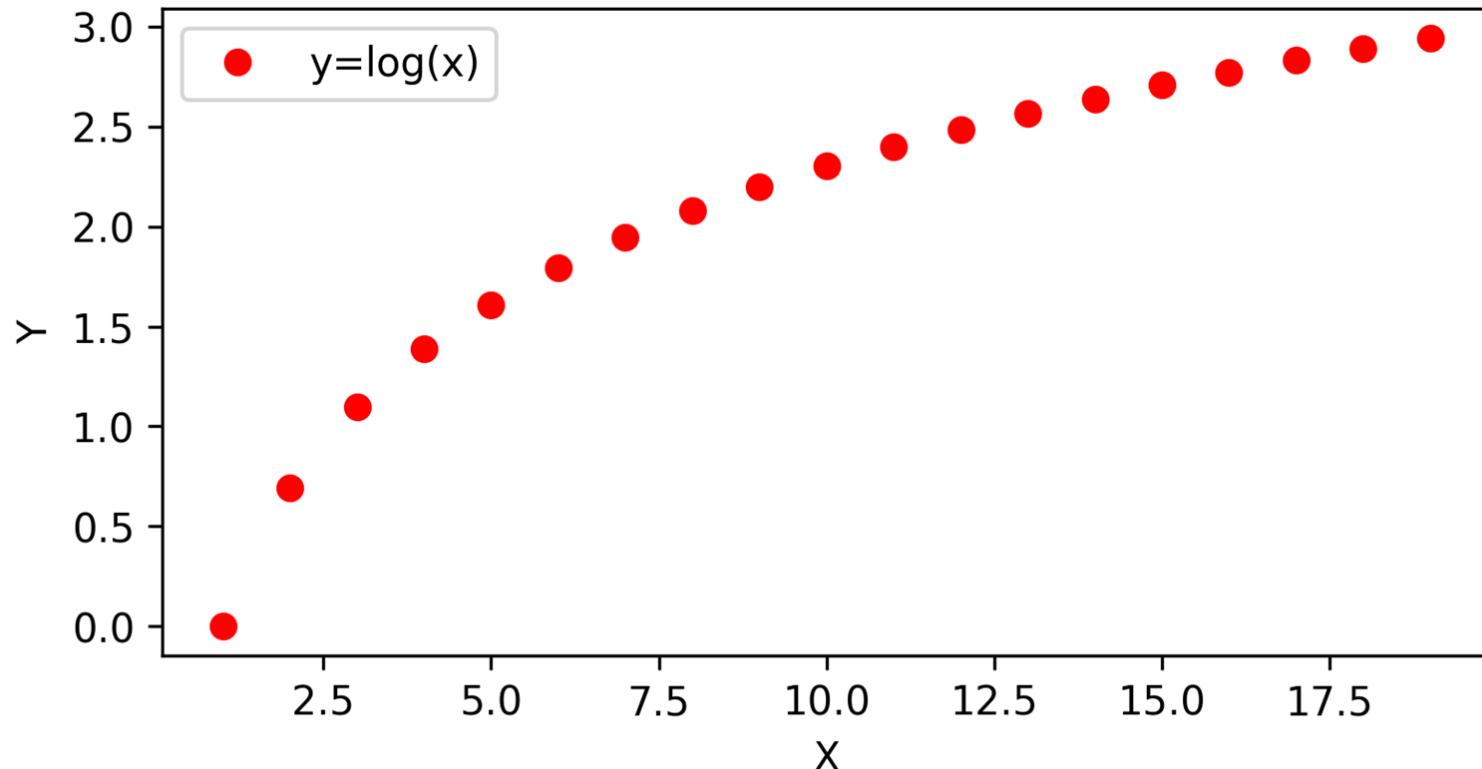
- We just completed a Linear Regression task, allowing us to predict future label values given a set of features!
- How can we now improve on a Linear Regression model?
- One approach is to consider **higher order relationships** on the features.

Polynomial Regression

- There are two issues polynomial regression will address for us:
 - Non-linear feature relationships to label
 - Interaction terms between features
- Let's first explore non-linear relationships and how considering polynomial orders could help address this.

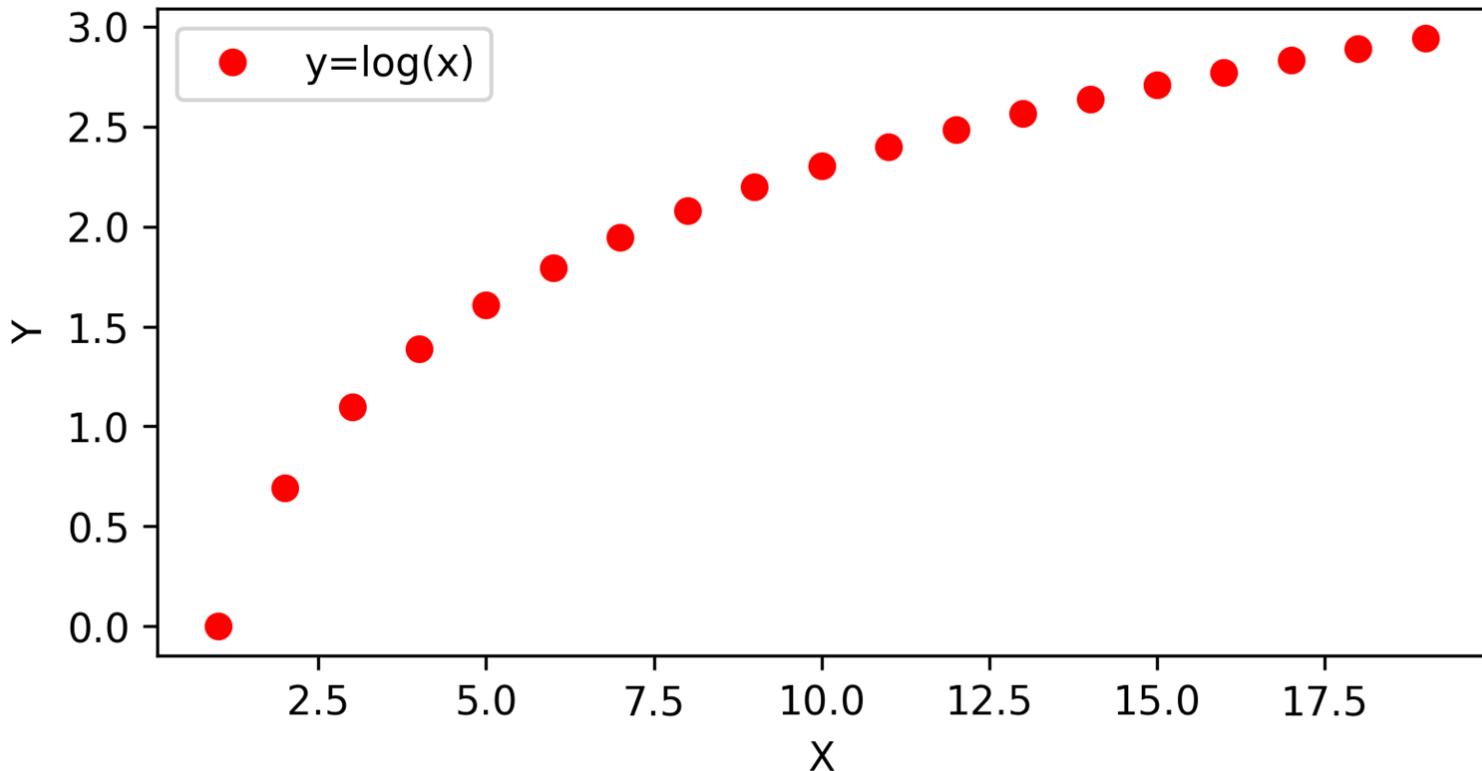
Polynomial Regression

- We know $\log(x)$ is not a linear relationship.



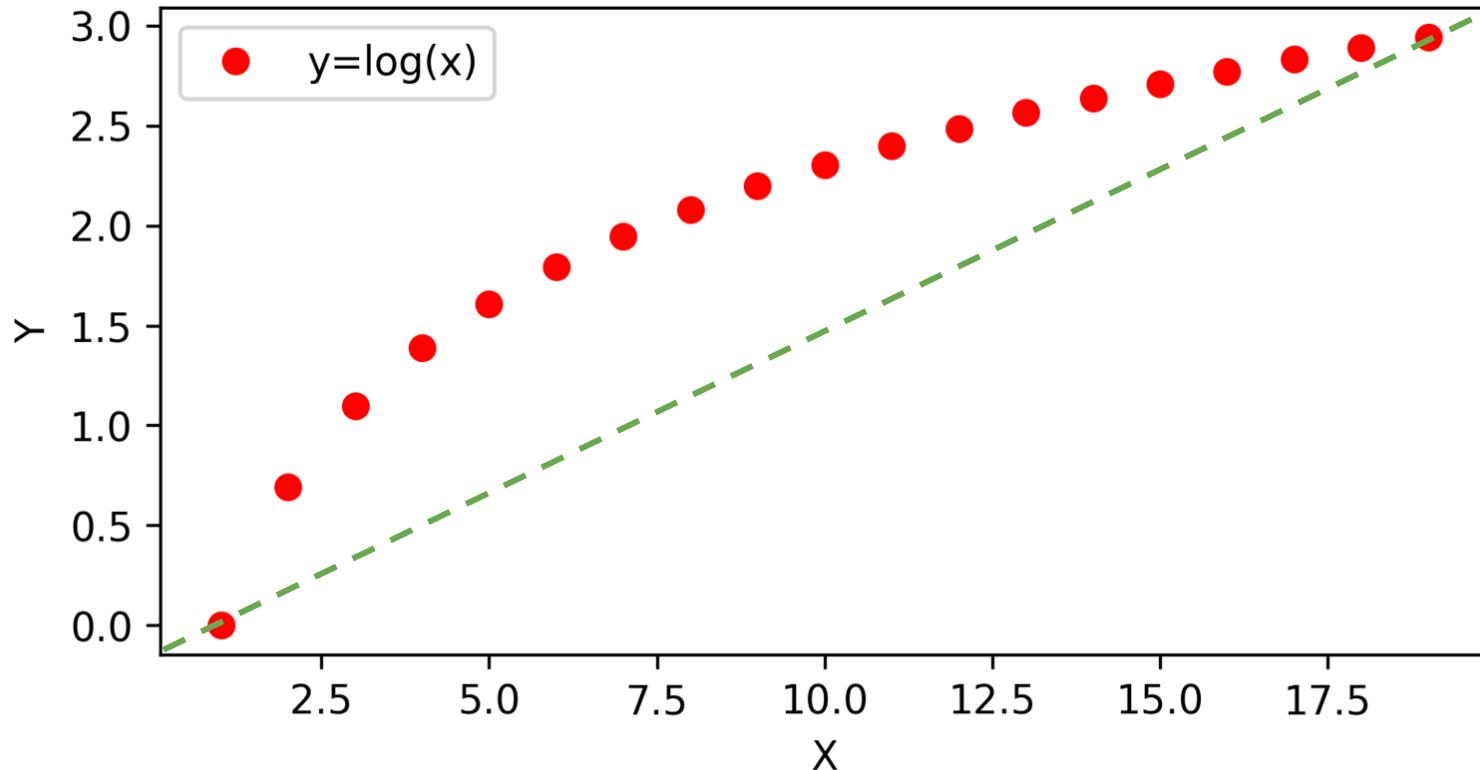
Polynomial Regression

- What is a feature X behaved like $\log(x)$?



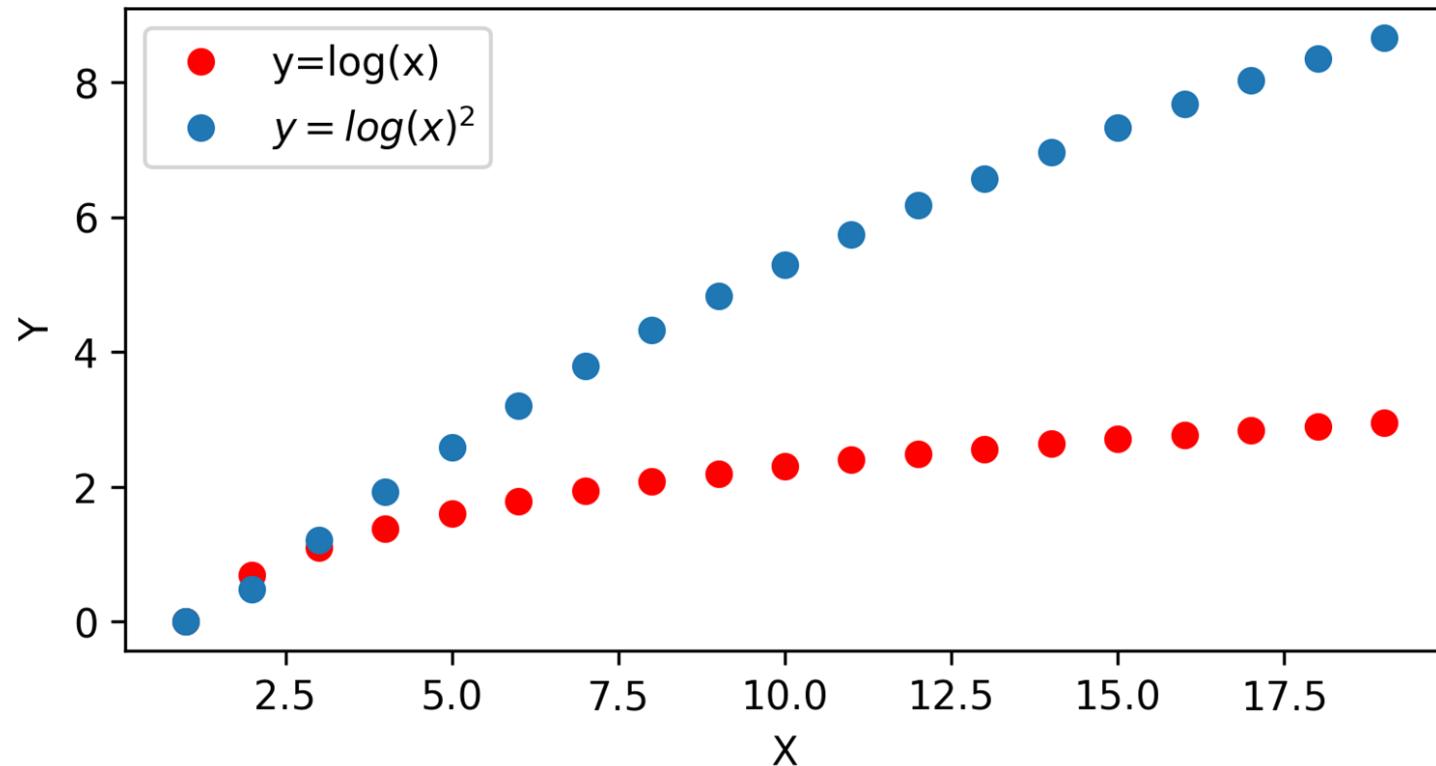
Polynomial Regression

- Will be difficult to find a linear relationship



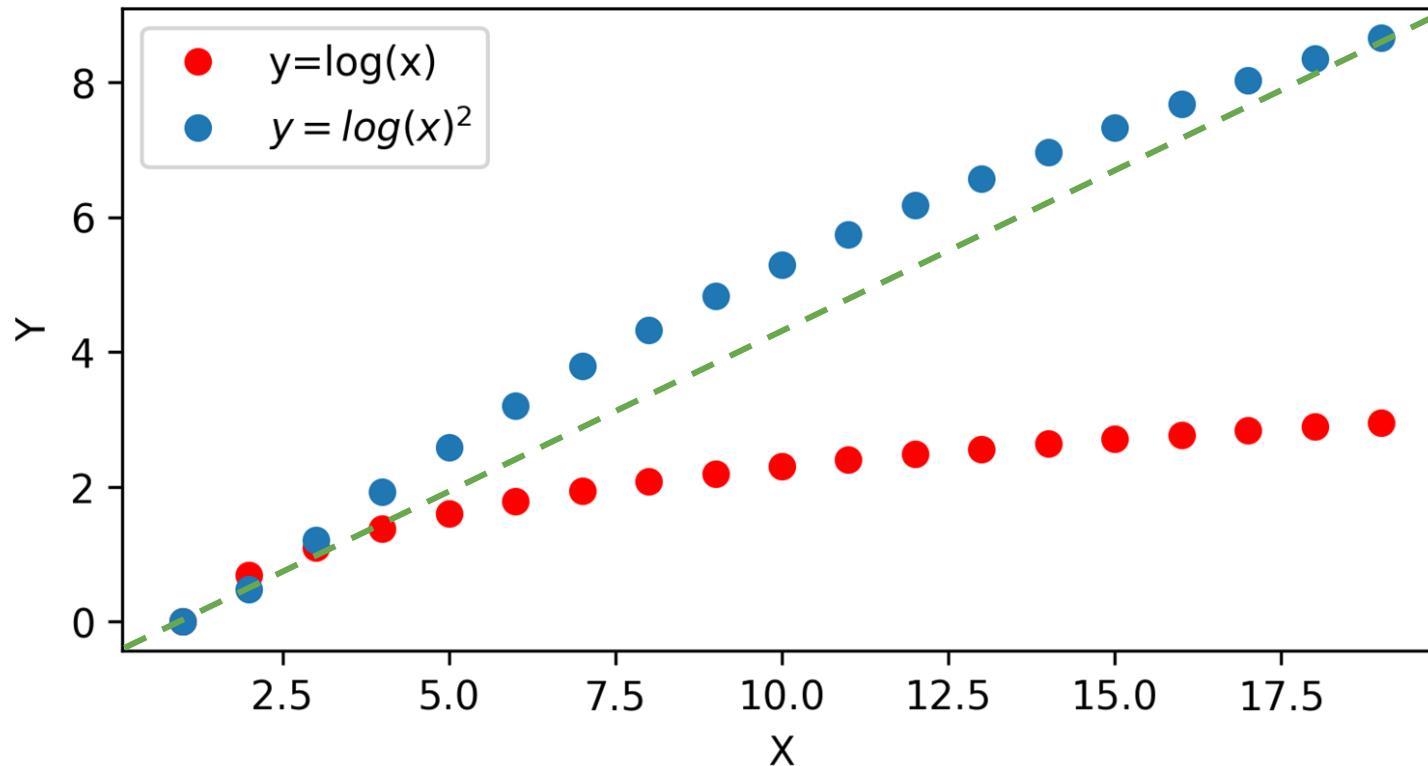
Polynomial Regression

- What about the square of this feature?



Polynomial Regression

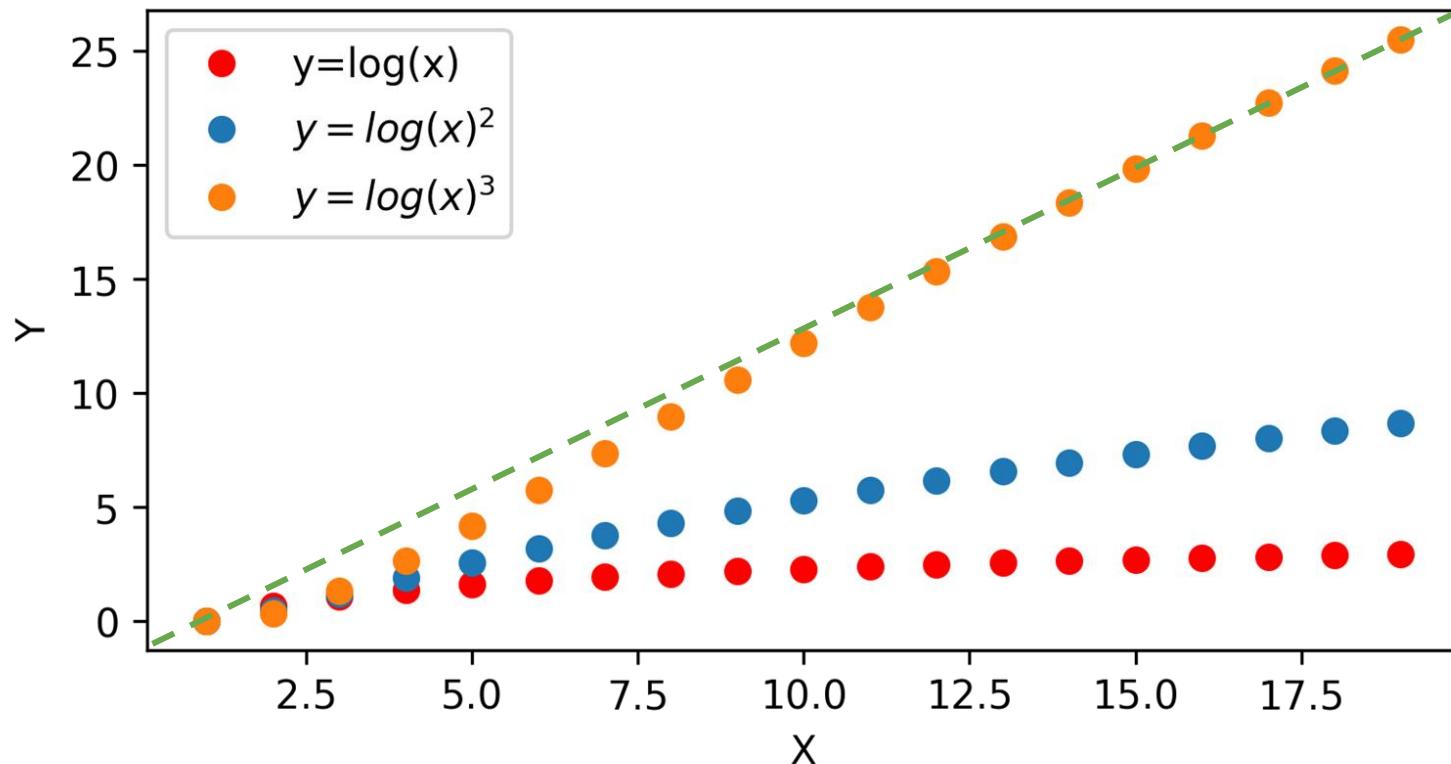
- Relationship could be more linear.





Polynomial Regression

- Even more so for higher orders!



Polynomial Regression

- Keep in mind this is an exaggerated example, and not every feature will have relationships at a higher order.
- The main point here is to show it could be reasonable to solve for a single linear Beta coefficient for polynomial of an original feature.

Polynomial Regression

- Let's now also consider **interaction terms**.
- What if features are only significant when in sync with one another?
- For example:
 - Perhaps newspaper advertising spend by itself is not effective, but greatly increases effectiveness if added to a TV advertising campaign.

Polynomial Regression

- Consumers only watching a TV ad will create some sales, but consumers who watch TV **and** are later “reminded” through a newspaper ad could contribute even more sales than TV or newspaper alone!
- How can we check for this?

Polynomial Regression

- Simplest way is to create a new feature that multiplies two existing features together to create an **interaction term**.
- We can keep the original features, and add on this **interaction term**.
- Fortunately Scikit-Learn does this for us easily through a **preprocessing** call.

Polynomial Regression

- Scikit-Learn's preprocessing library contains many useful tools to apply to the original data set **before** model training.
- One tool is the **PolynomialFeatures** which automatically creates both higher order feature polynomials and the interaction terms between all feature combinations.

Polynomial Regression

- The features created include:
 - The bias (the value of 1.0)
 - Values raised to a power for each degree (e.g. x^1, x^2, x^3, \dots)
 - Interactions between all pairs of features (e.g. $x_1 * x_2, x_1 * x_3, \dots$)

Polynomial Regression

- Converting Two Features **A** and **B**
 - **1, A, B, A², AB, B²**

Polynomial Regression

- Converting Two Features **A** and **B**
 - **1, A, B, A², AB, B²**
- Generalized terms of features **X₁** and **X₂**
 - **1, X₁, X₂, X₁², X₁X₂, X₂²**

Polynomial Regression

- Converting Two Features **A** and **B**
 - **1, A, B, A², AB, B²**
- Generalized terms of features **X₁** and **X₂**
 - **1, X₁, X₂, X₁², X₁X₂, X₂²**
- Example if row was **X₁=2** and **X₂=3**
 - **1, 2, 3, 4, 6, 9**

Bias-Variance Trade Off

Overfitting versus Underfitting

Overfitting and Underfitting

- In general, increasing model complexity in search for better performance leads to a **Bias-Variance trade-off**.
- We want to have a model that can generalize well to new unseen data, but can also account for variance and patterns in the known data.

Overfitting and Underfitting

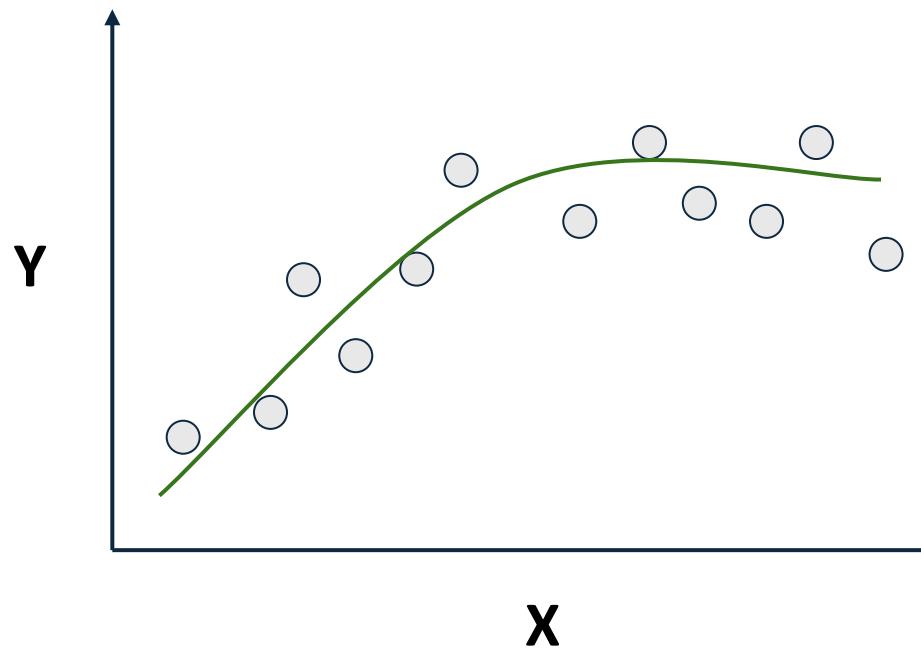
- Extreme bias or extreme variance both lead to bad models.
- We can visualize this effect by considering a model that underfits (high bias) or a model that overfits (high variance).
- Let's start with a model that overfits to a dataset...

Overfitting and Underfitting

- **Overfitting**
 - The model fits too much to the noise from the data.
 - This often results in **low error on training sets but high error on test/validation sets.**

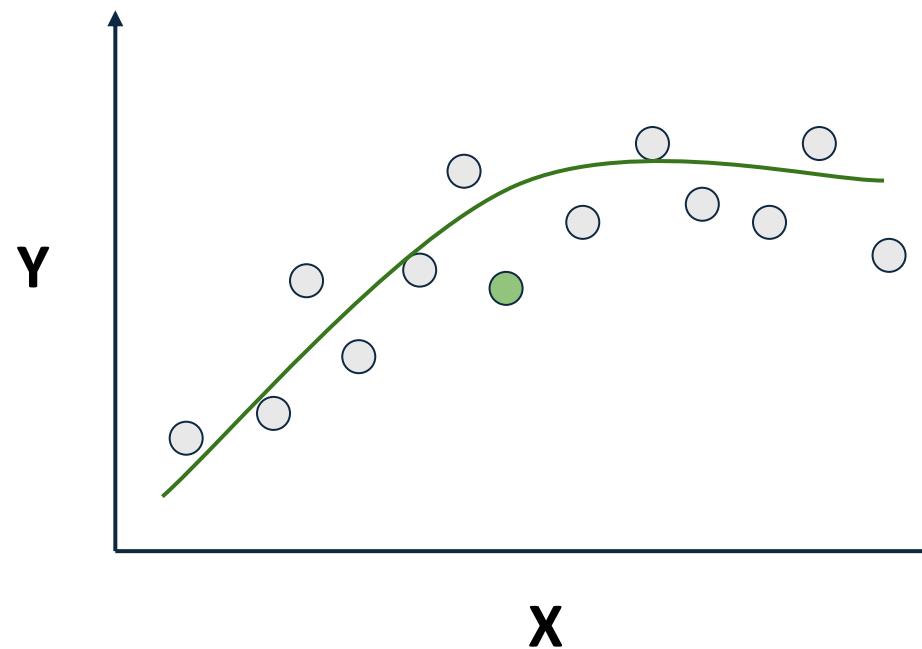
Overfitting and Underfitting

Good Model



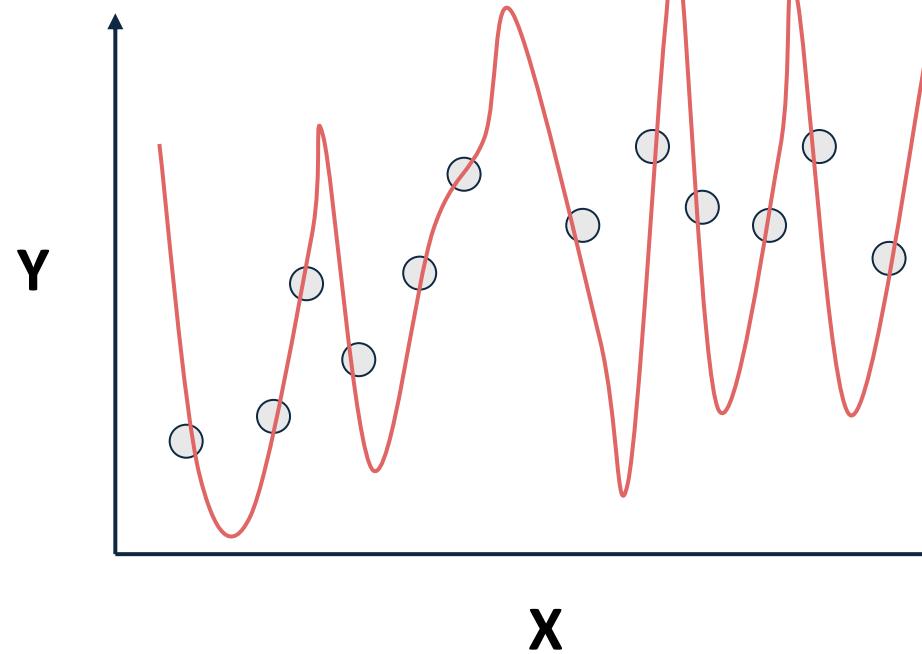
Overfitting and Underfitting

Good Model



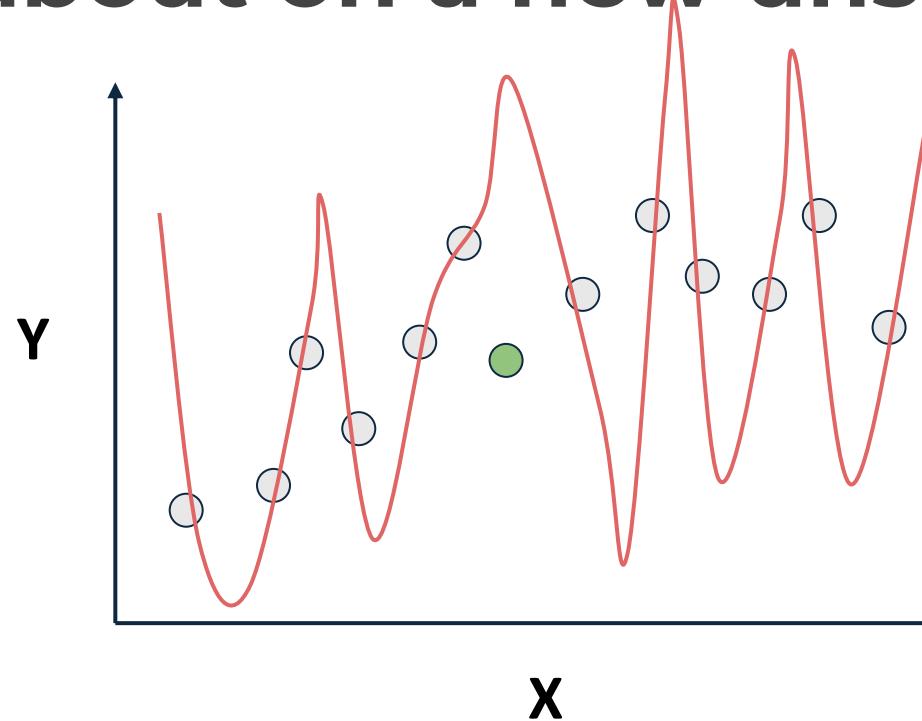
Overfitting and Underfitting

- **Overfitting**



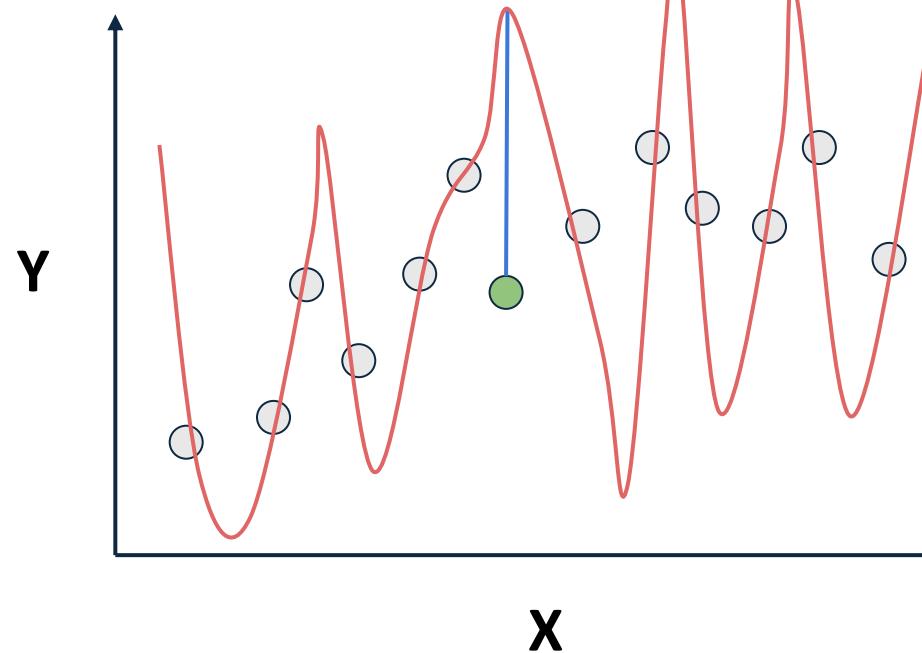
Overfitting and Underfitting

- **But what about on a new unseen data point?**



Overfitting and Underfitting

- **Overfitting can cause large test errors!**



Overfitting and Underfitting

- **Overfitting**
 - Model is fitting too much to noise and variance in the training data.
 - Model will perform very well on training data, but have poor performance on new unseen data.

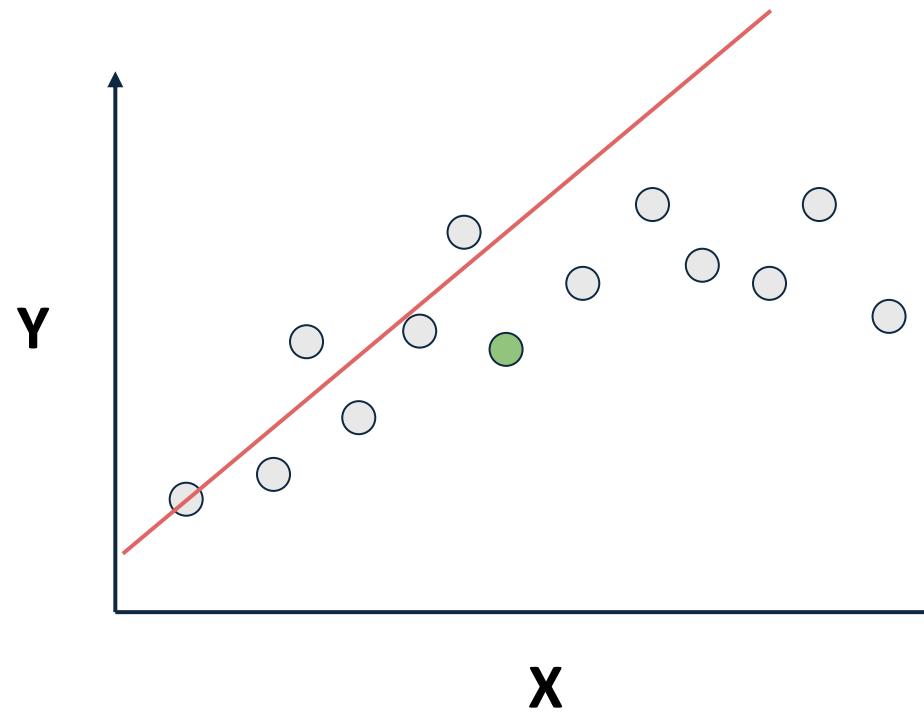
Overfitting and Underfitting

- **Underfitting**

- Model does not capture the underlying trend of the data and does not fit the data well enough.
- Low variance but high bias.
- Underfitting is often a result of an excessively simple model.

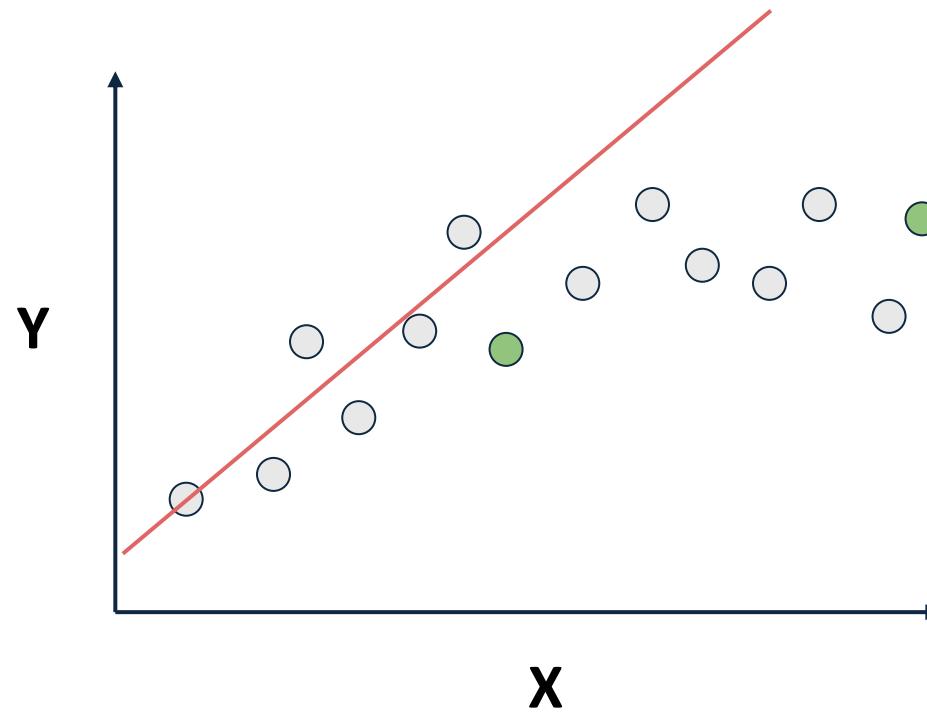
Overfitting and Underfitting

Underfitting



Overfitting and Underfitting

Underfitting



Overfitting and Underfitting

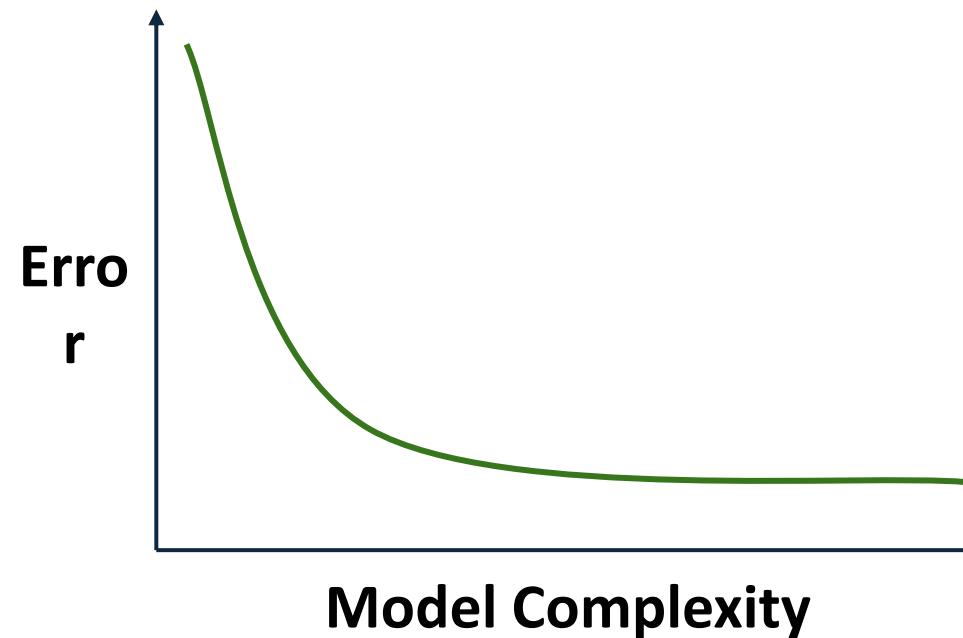
- **Underfitting**
 - Model has high bias and is generalizing too much.
 - Underfitting can lead to poor performance in both training and testing data sets.

Overfitting and Underfitting

- **Overfitting versus Underfitting**
 - Overfitting can be harder to detect, since good performance on training data could lead to a model that appears to be performing well.

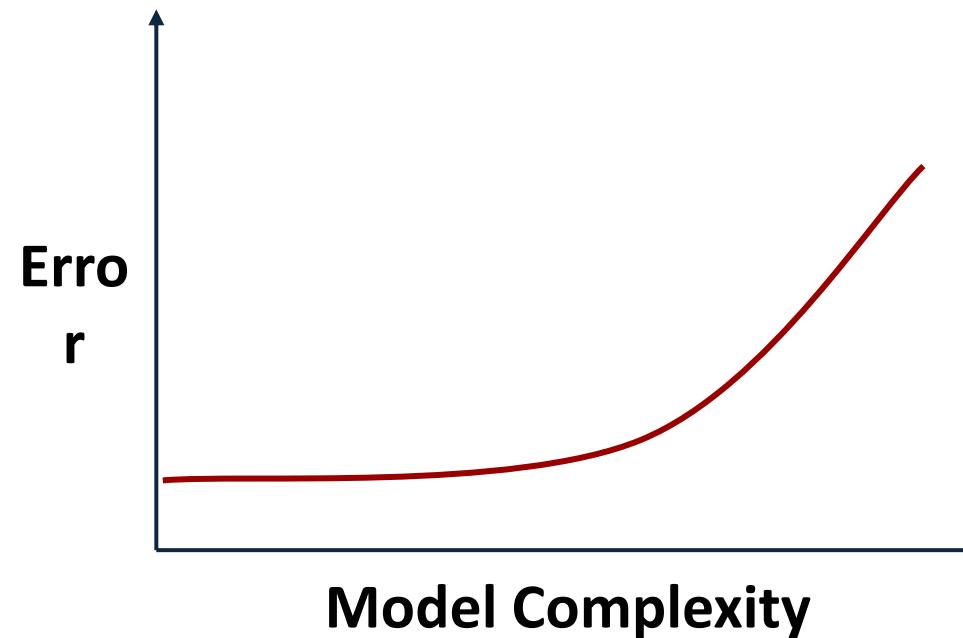
Overfitting and Underfitting

- Good Model



Overfitting and Underfitting

- Bad Model



Overfitting and Underfitting

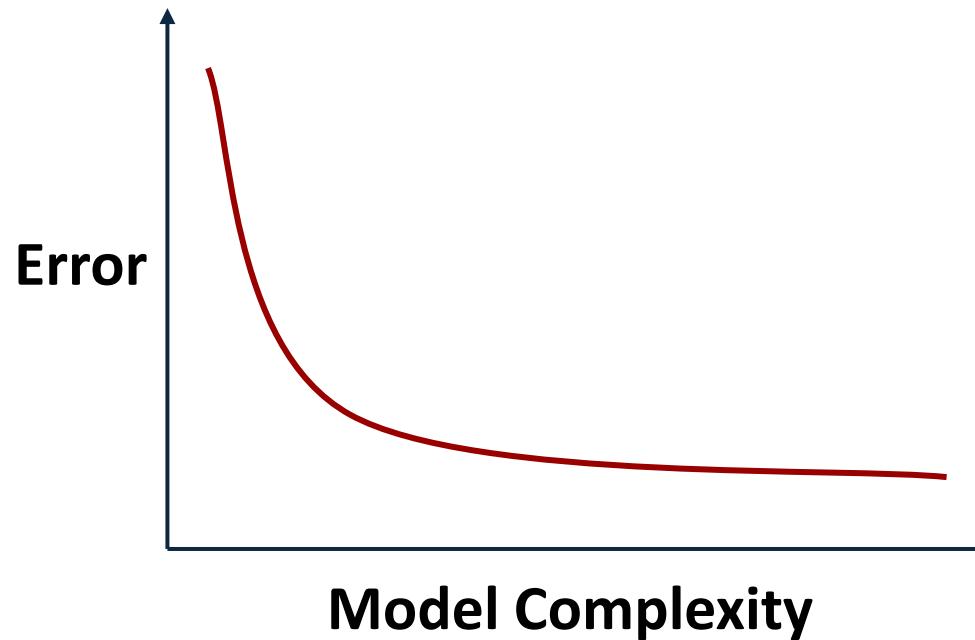
- When thinking about **overfitting** and **underfitting** we want to keep in mind the relationship of model performance on the training set versus the test/validation set.

Overfitting and Underfitting

- Let's imagine we split our data into a **training set** and a **test set**

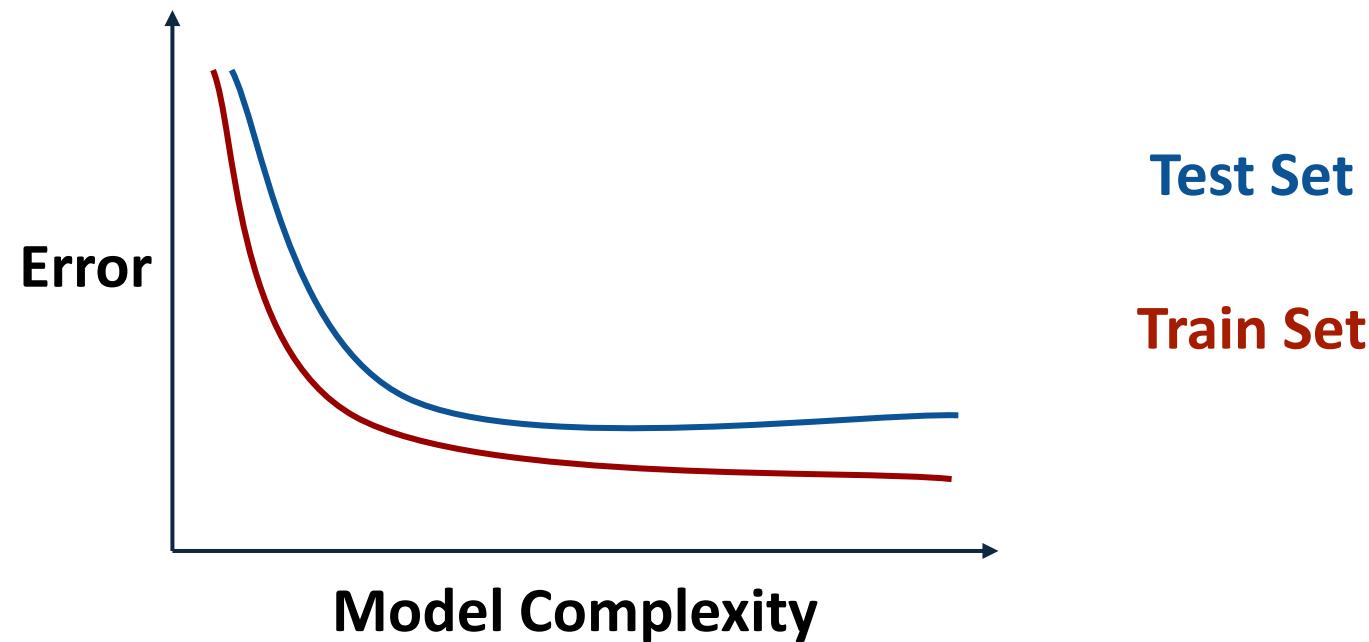
Overfitting and Underfitting

- We first see performance on the **training set**



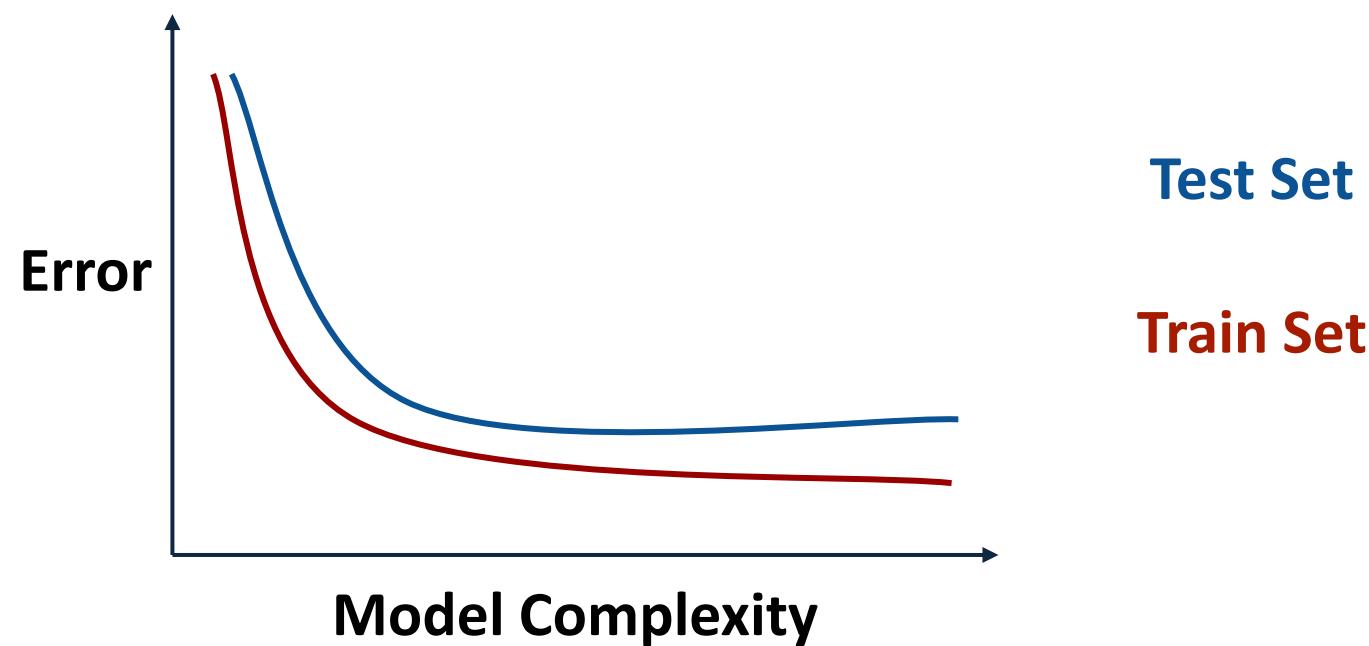
Overfitting and Underfitting

- Next we check performance on the **test set**



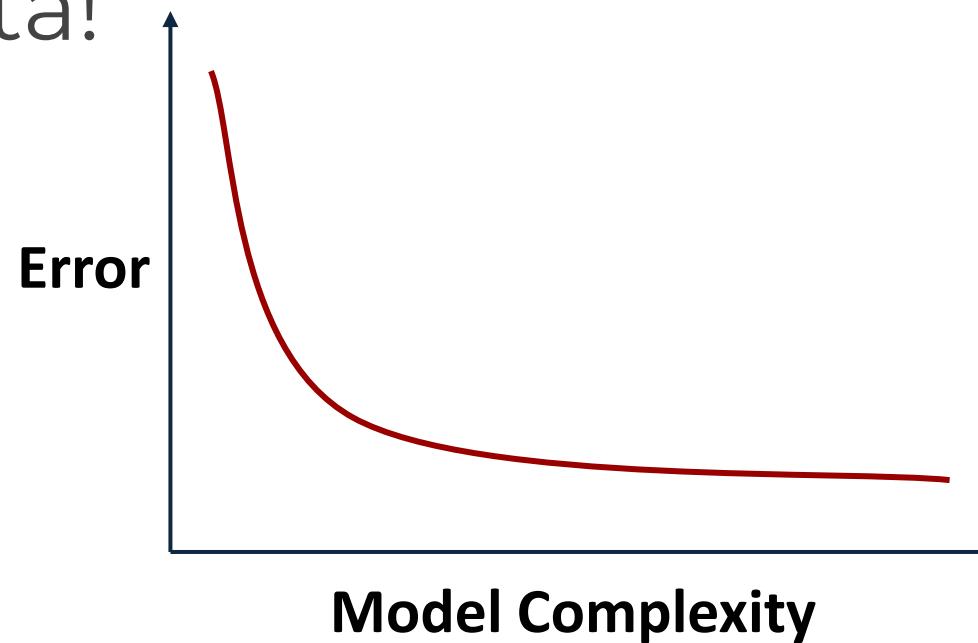
Overfitting and Underfitting

- Ideally the model would perform well on both, with similar behavior.



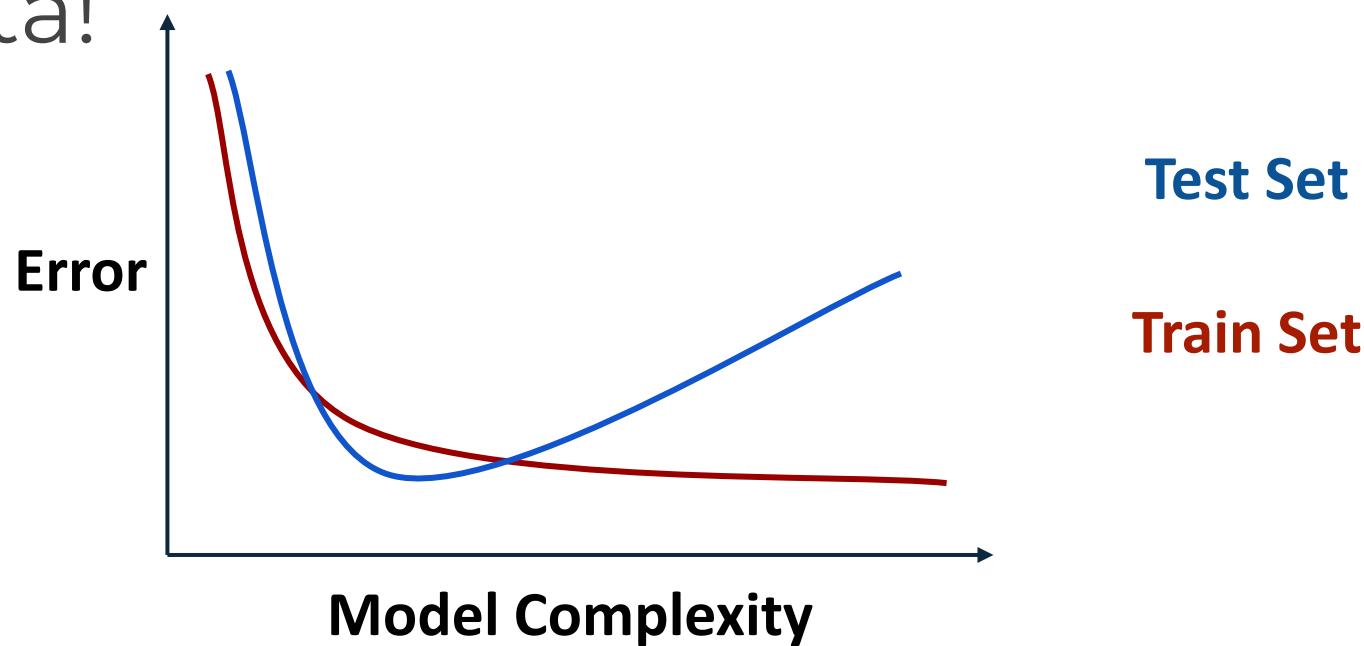
Overfitting and Underfitting

- But what happens if we overfit on the training data? That means we would perform poorly on new test data!



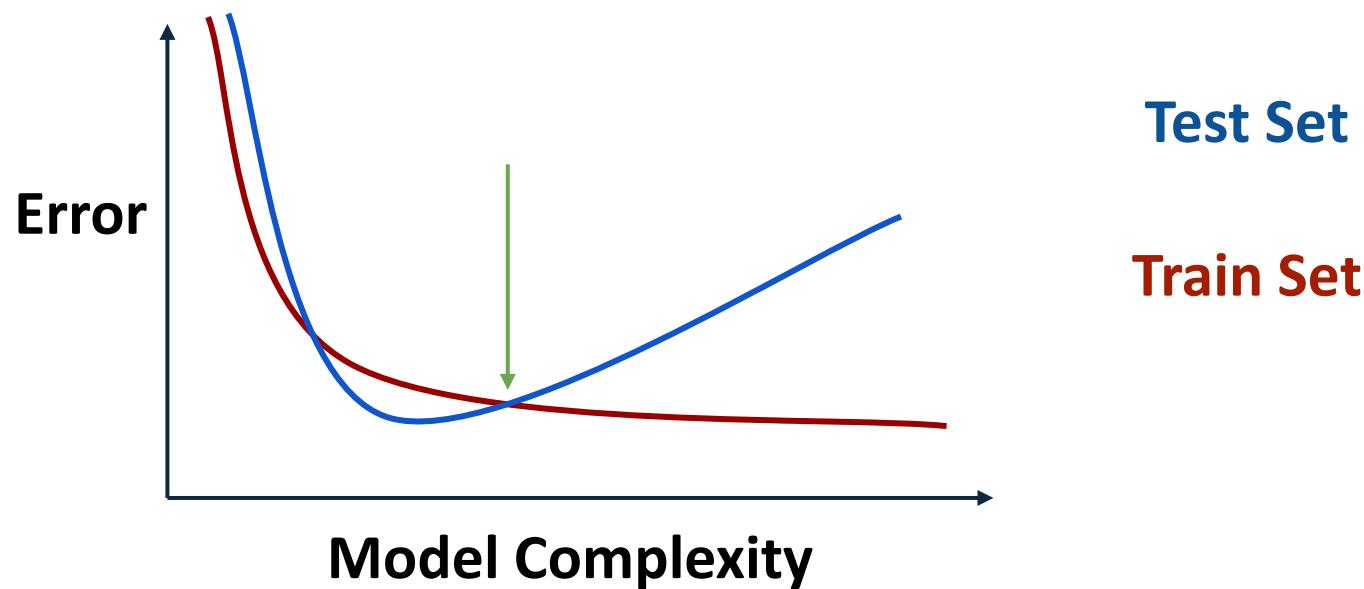
Overfitting and Underfitting

- But what happens if we overfit on the training data? That means we would perform poorly on new test data!



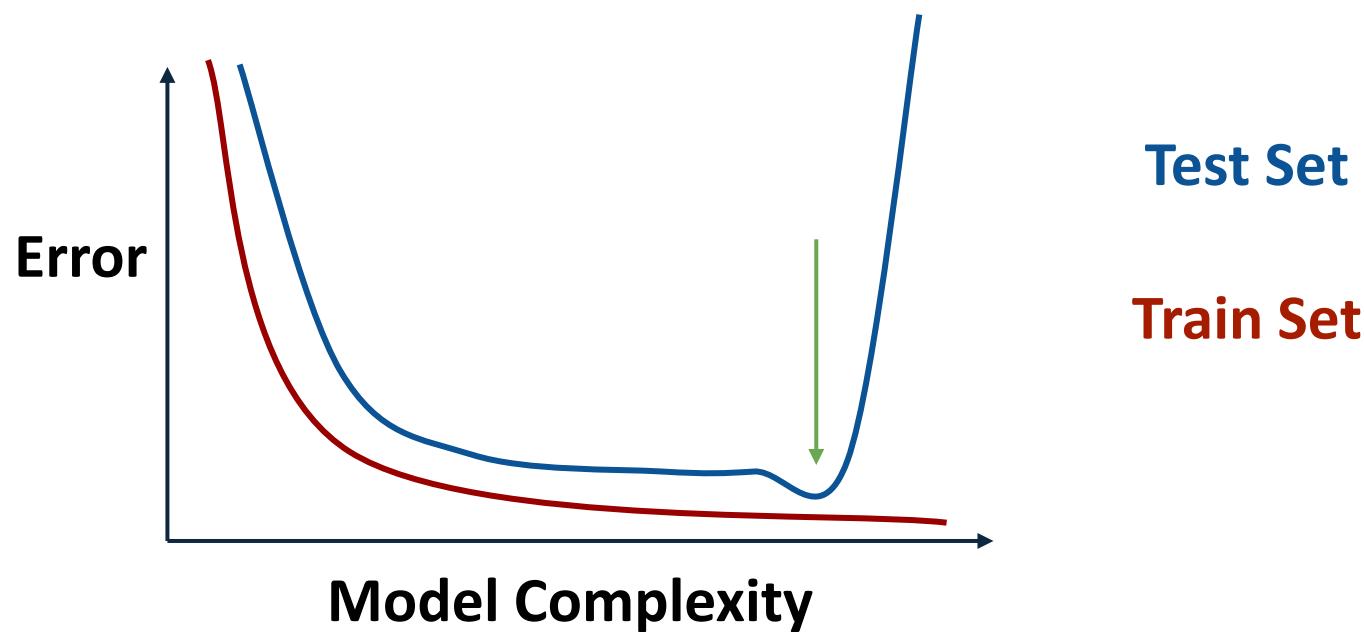
Overfitting and Underfitting

- This is a good indication too much complexity, you should look for the point to determine appropriate values!



Overfitting and Underfitting

- For certain algorithms this test error jump can be sudden instead of gradual.



Overfitting and Underfitting

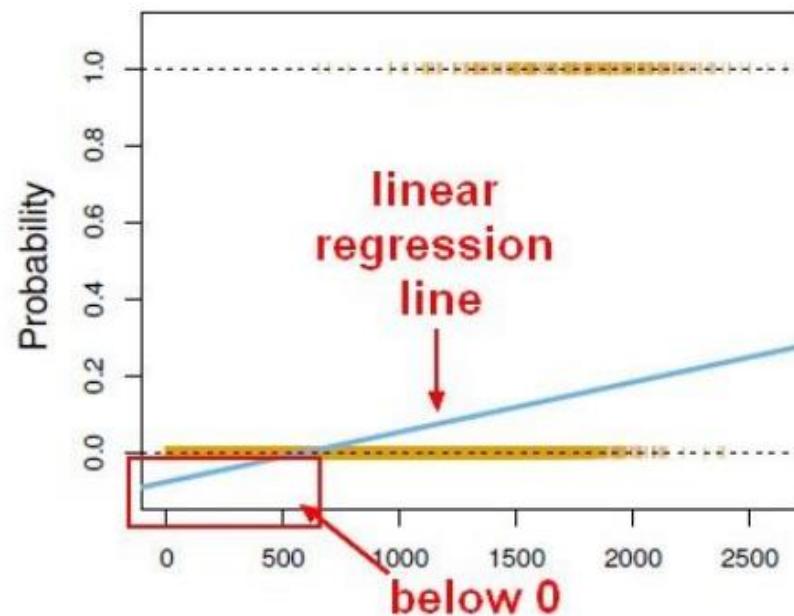
- This means when deciding optimal model complexity **and** wanting to fairly evaluate our model's performance, we can consider both the train error and test error to select an ideal complexity.

Overfitting and Underfitting

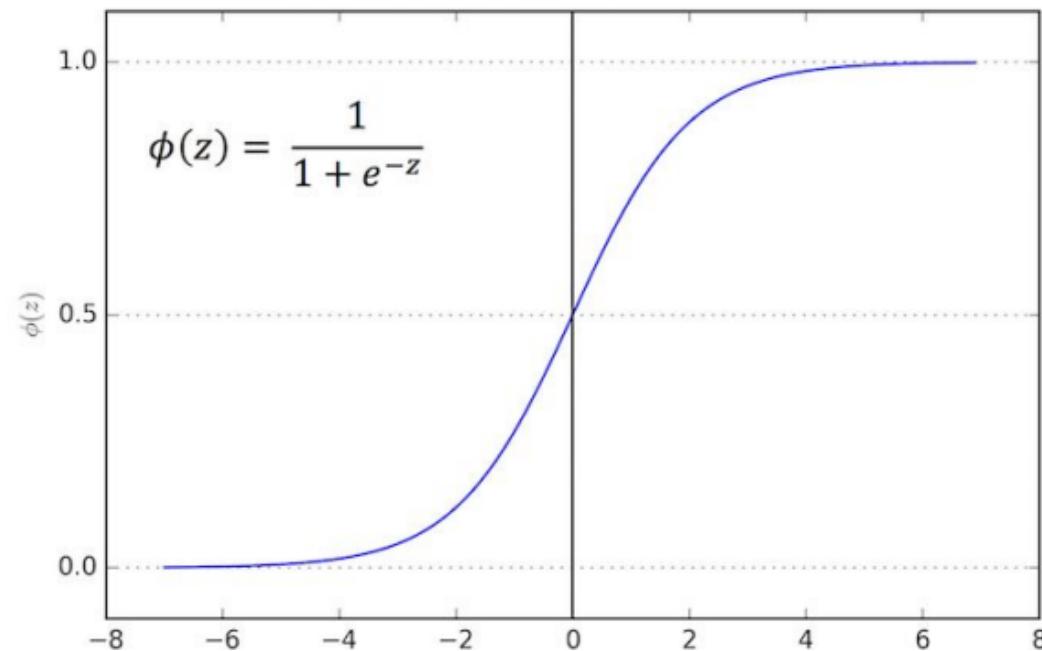
- In the case of Polynomial Regression, complexity directly relates to degree of the polynomial, but many machine learning algorithms have their own hyperparameters that can increase complexity.

- We want to learn about Logistic Regression as a method for **Classification**.
- Some examples of classification problems:
 - Spam versus “Ham” emails
 - Loan Default (yes/no)
 - Disease Diagnosis
- Above were all examples of Binary Classification

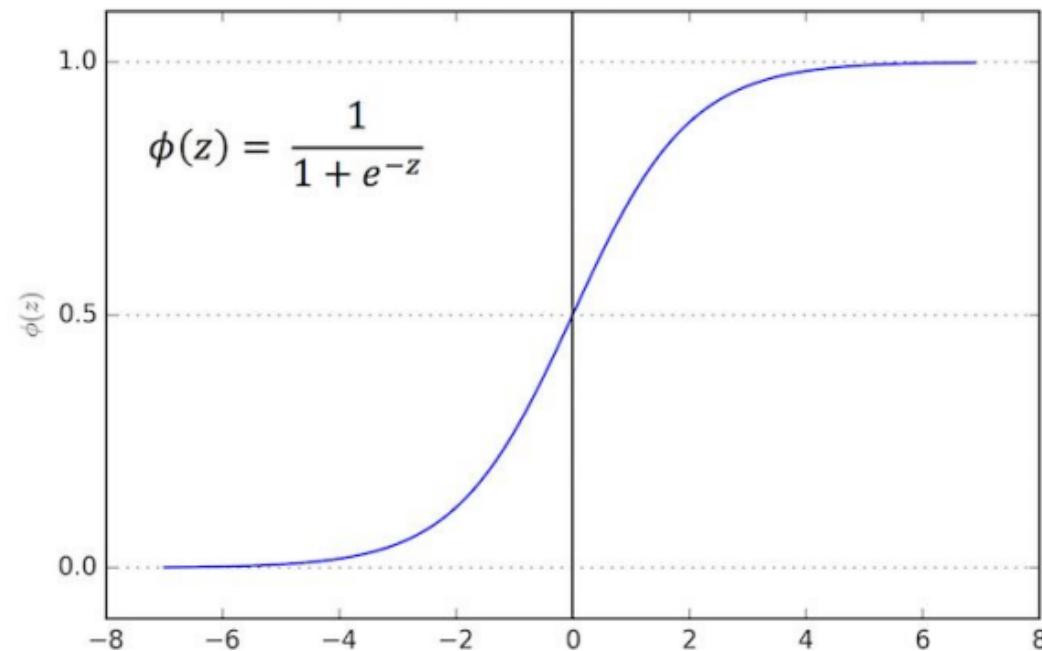
- We can't use a normal linear regression model on binary groups. It won't lead to a good fit:



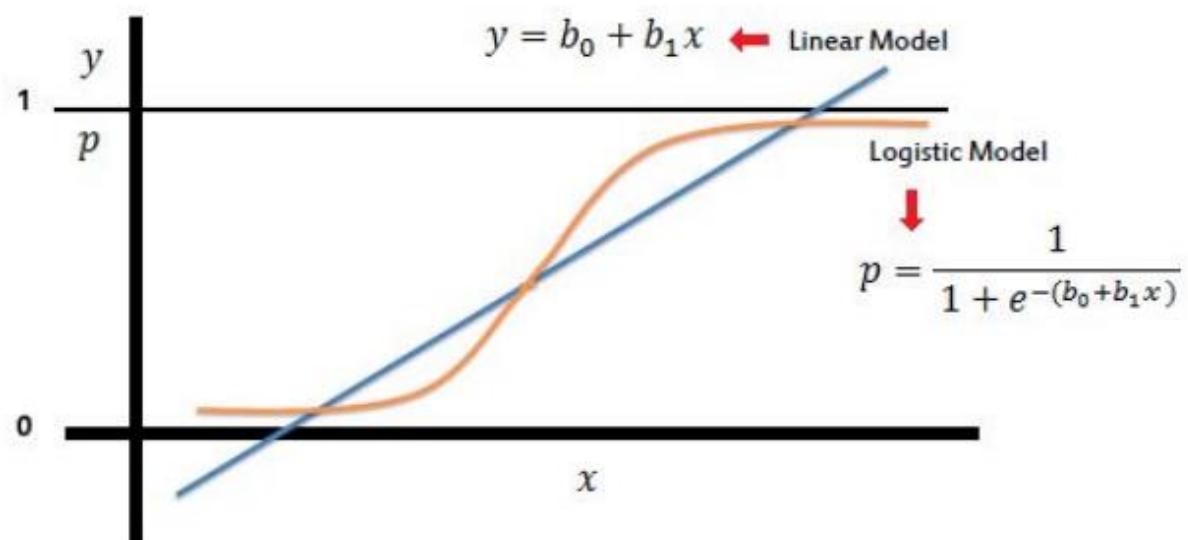
- The Sigmoid (aka Logistic) Function takes in any value and outputs it to be between 0 and 1.



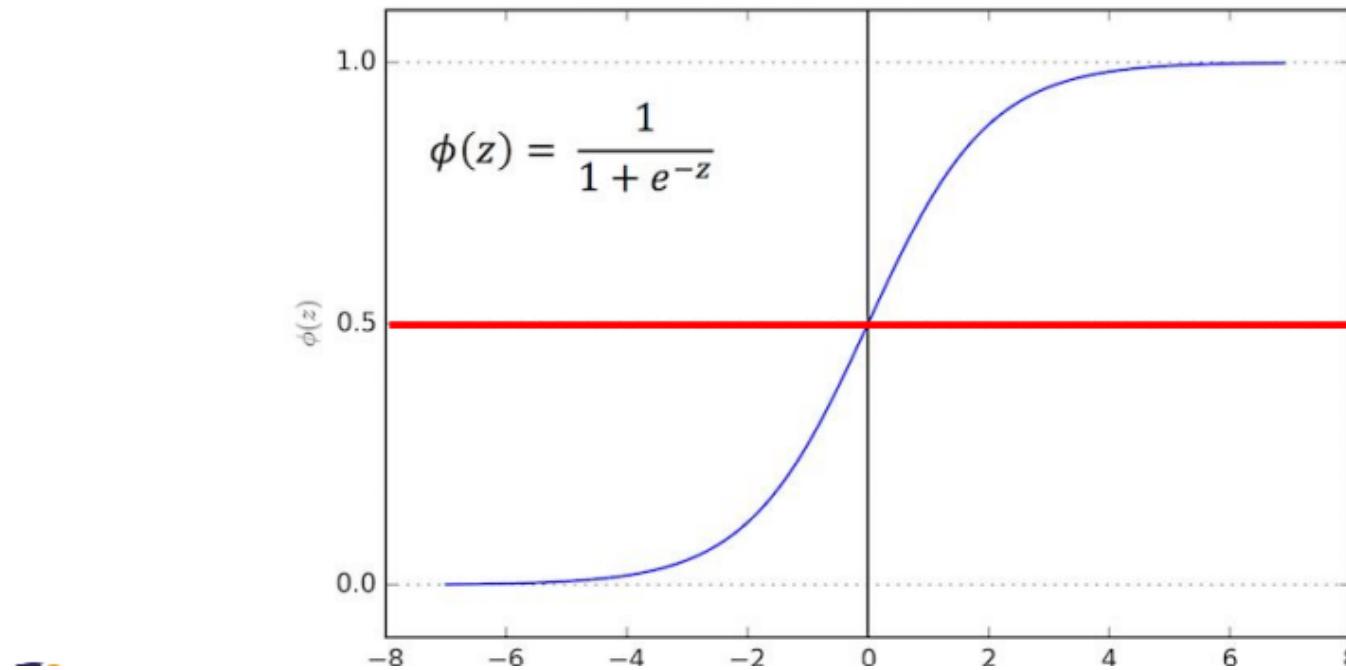
- The Sigmoid (aka Logistic) Function takes in any value and outputs it to be between 0 and 1.



- This means we can take our Linear Regression Solution and place it into the Sigmoid Function.



- We use the logistic function to output a value ranging from 0 to 1. Based off of this probability we assign a class.



n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

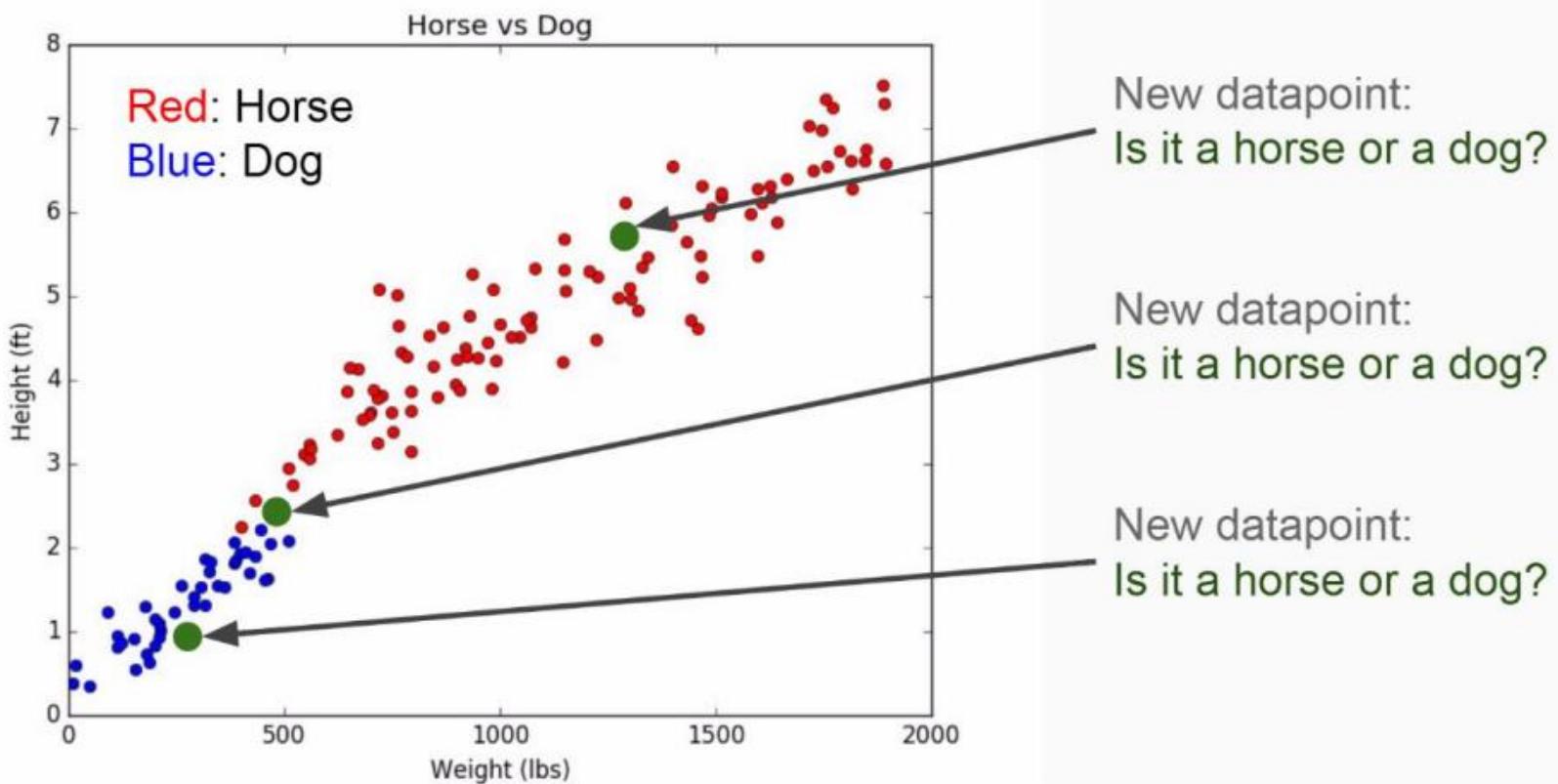
Accuracy:

- Overall, how often is it **correct?**
- $(TP + TN) / \text{total} = 150/165 = 0.91$

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Misclassification Rate (Error Rate):

- Overall, how often is it **wrong?**
- $(FP + FN) / \text{total} = 15/165 = 0.09$



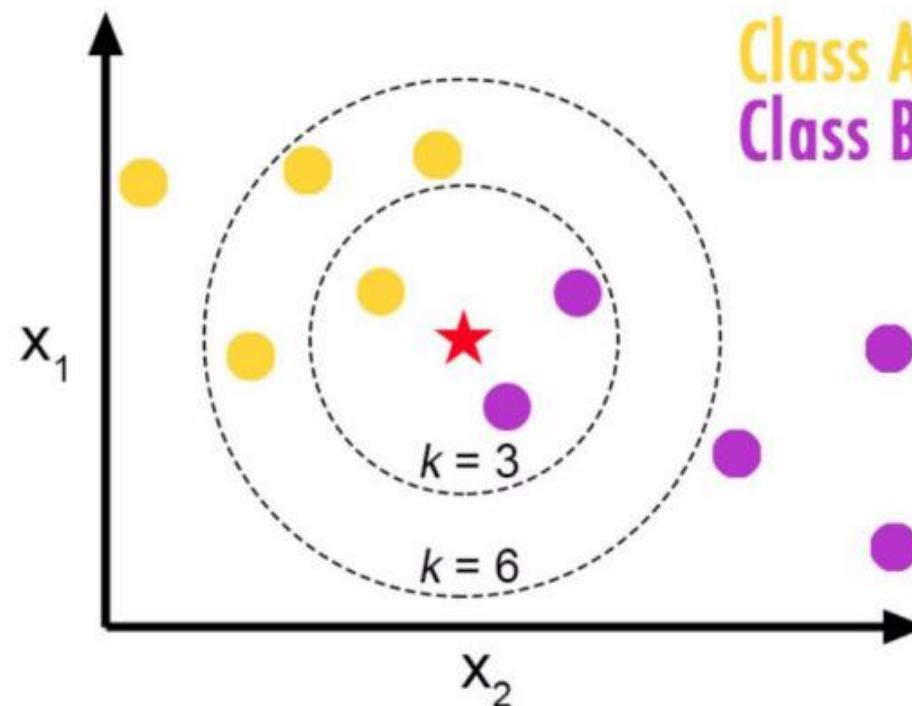
Training Algorithm:

1. Store all the Data

Prediction Algorithm:

1. Calculate the distance from x to all points in your data
2. Sort the points in your data by increasing distance from x
3. Predict the majority label of the “ k ” closest points

Choosing a K will affect what class a new point is assigned to:



Decision Tree and Random Forest

Imagine that I play Tennis every Saturday and I always invite a friend to come with me.

Sometimes my friend shows up, sometimes not.

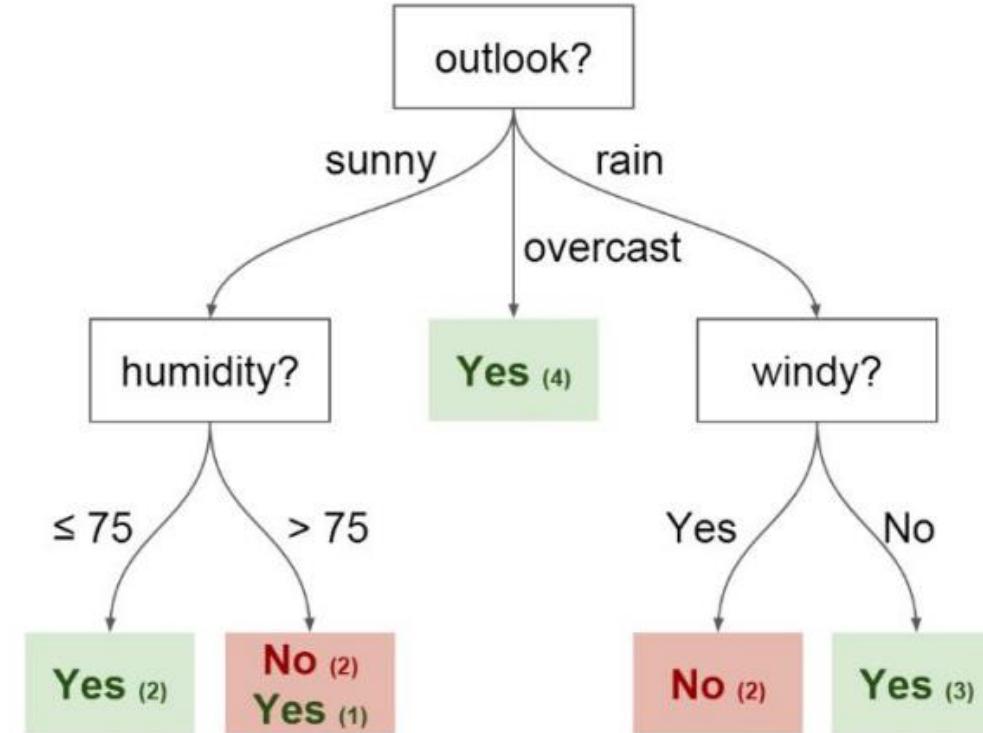
For him it depends on a variety of factors, such as: weather, temperature, humidity, wind etc..

I start keeping track of these features and whether or not he showed up to play with me.

Decision Tree and Random Forest

I want to use this data to predict whether or not he will show up to play.

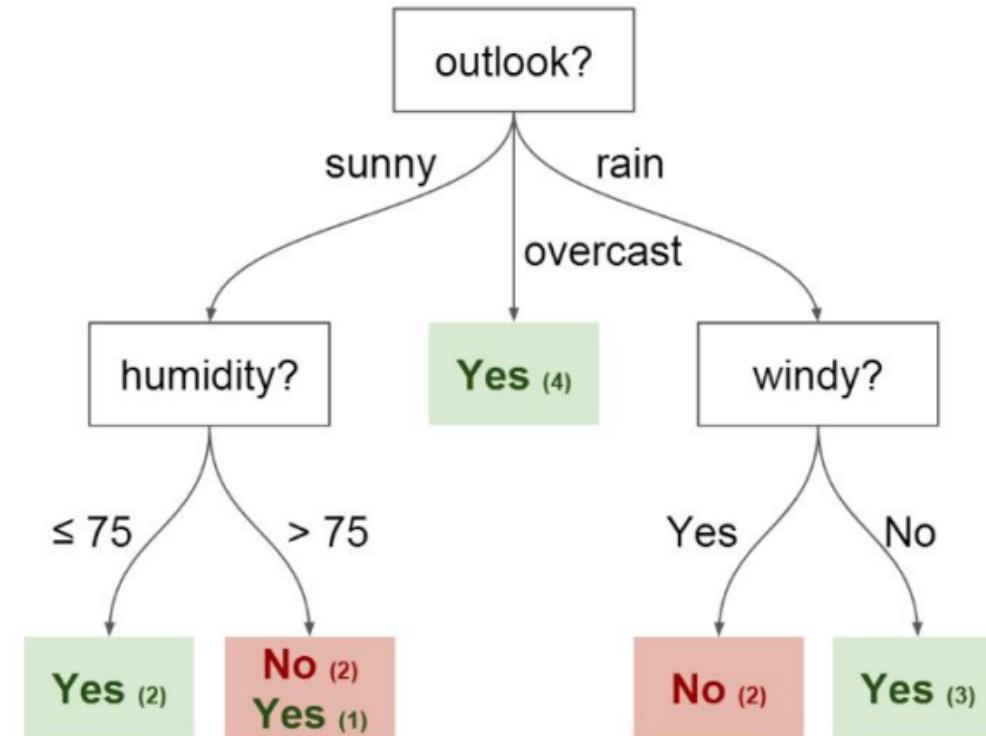
An intuitive way to do this is through a Decision Tree



Decision Tree and Random Forest

In this tree we have:

- Root
 - The node that performs the first split
- Leaves
 - Terminal nodes that predict the outcome

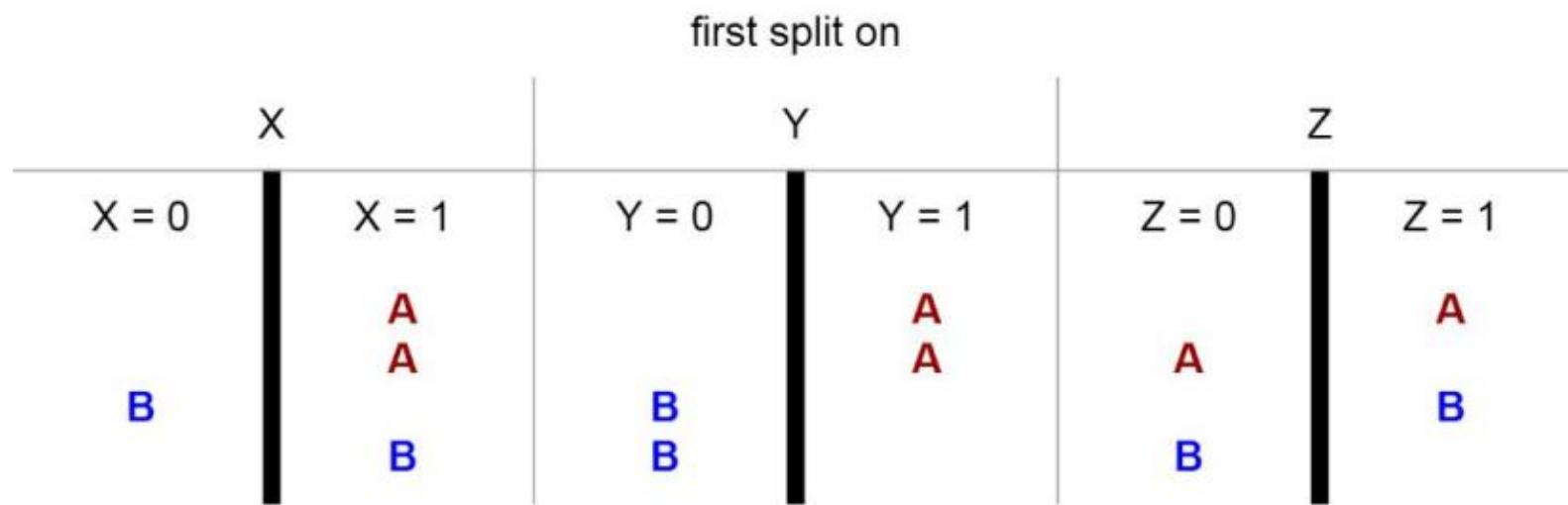


Decision Tree and Random Forest

Imaginary Data with 3 features (X,Y, and Z) with two possible classes.

X	Y	Z	Class
1	1	1	A
1	1	0	A
0	0	1	B
1	0	0	B

Decision Tree and Random Forest



Decision Tree and Random Forest

To improve performance, we can use many trees with a random sample of features chosen as the split.

- A new random sample of features is chosen for **every single tree at every single split.**
- For **classification**, m is typically chosen to be the square root of p .

Support Vector Machines

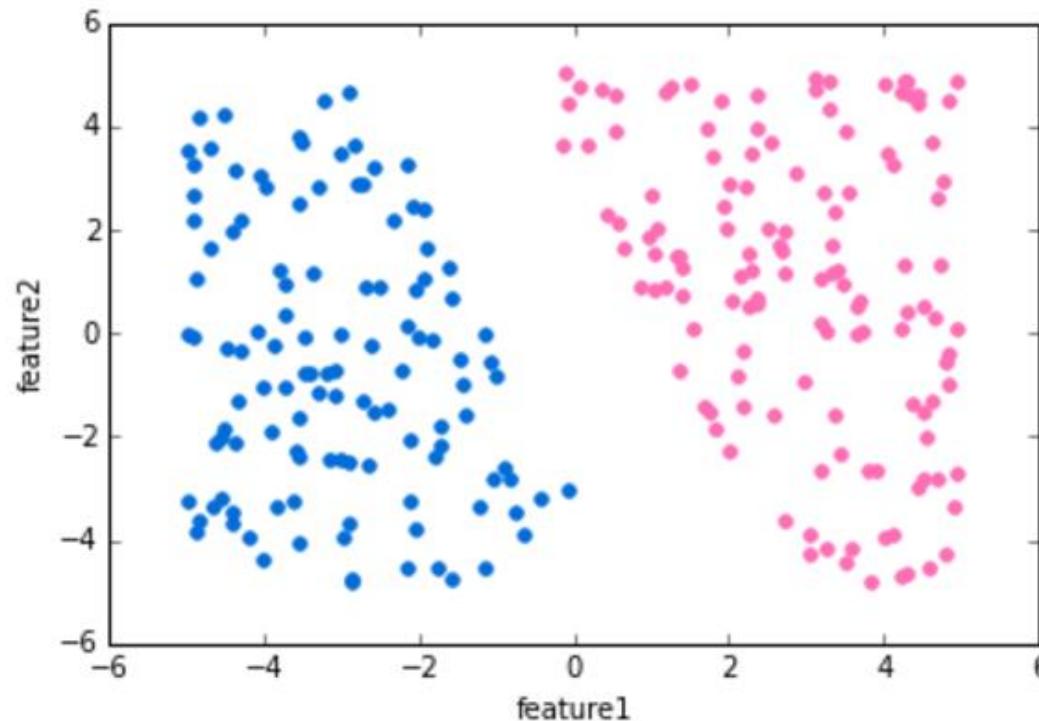
An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

Support Vector Machines

New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

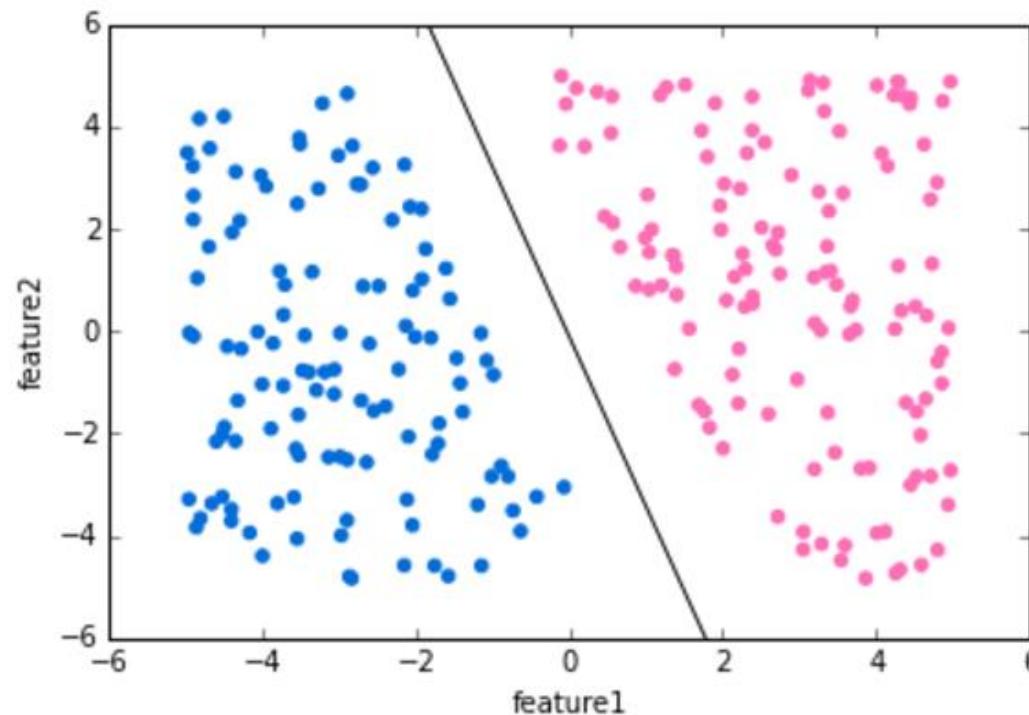
Support Vector Machines

Let's show the basic intuition behind SVMs. Imagine the labeled training data below:



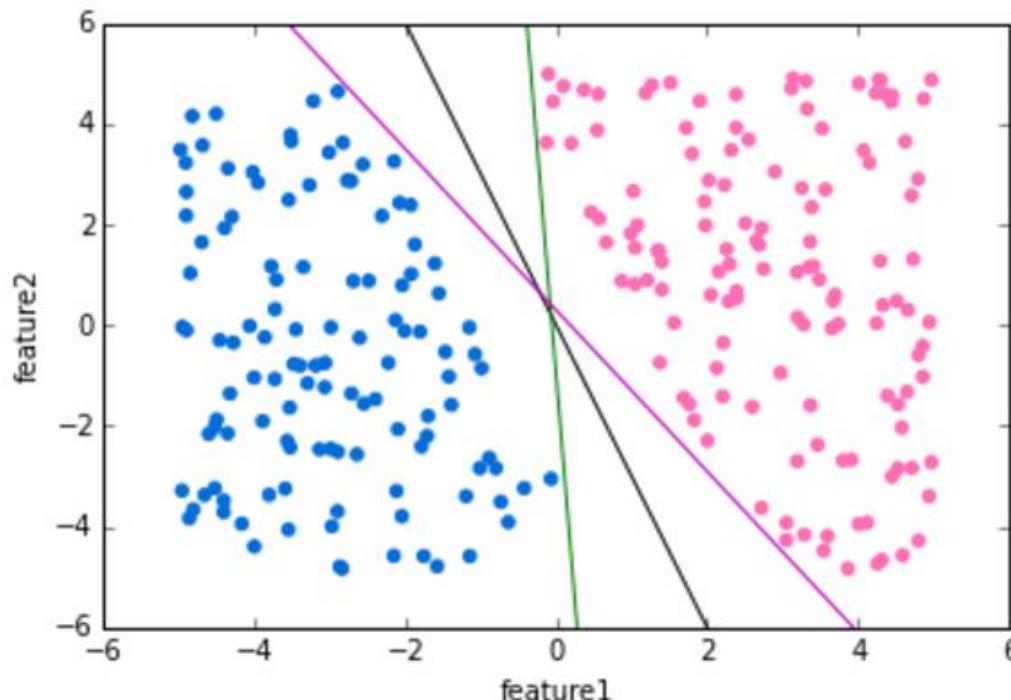
Support Vector Machines

We can draw a separating “hyperplane” between the classes.



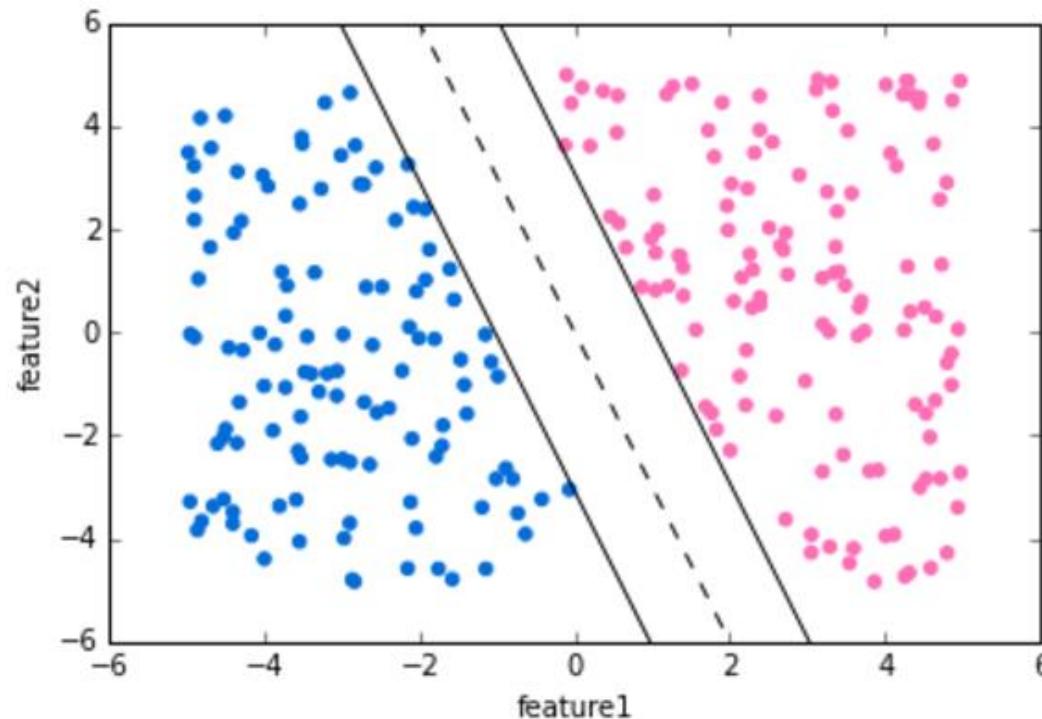
Support Vector Machines

But we have many options of hyperplanes that separate perfectly...



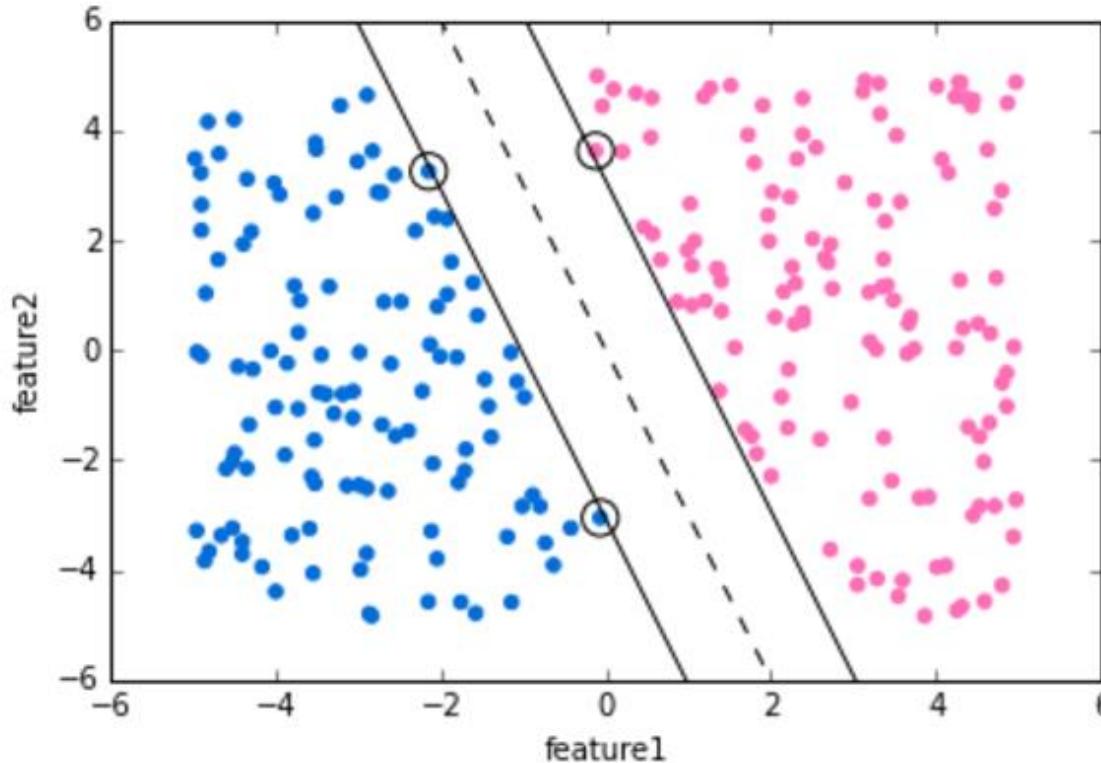
Support Vector Machines

We would like to choose a hyperplane that maximizes the margin between classes



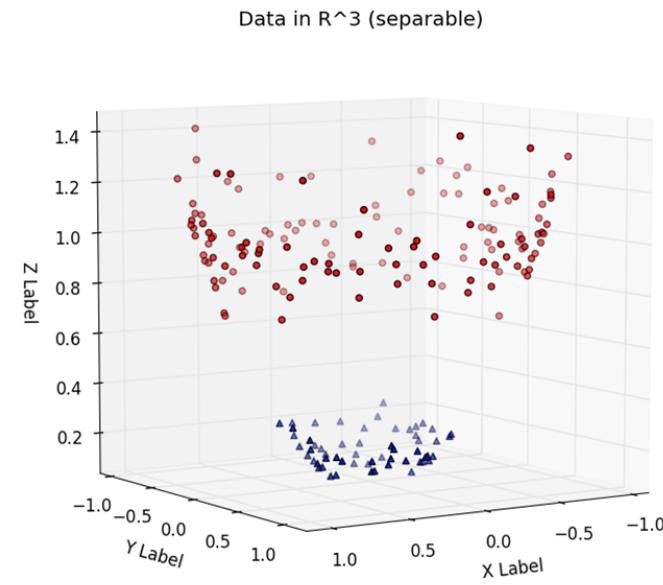
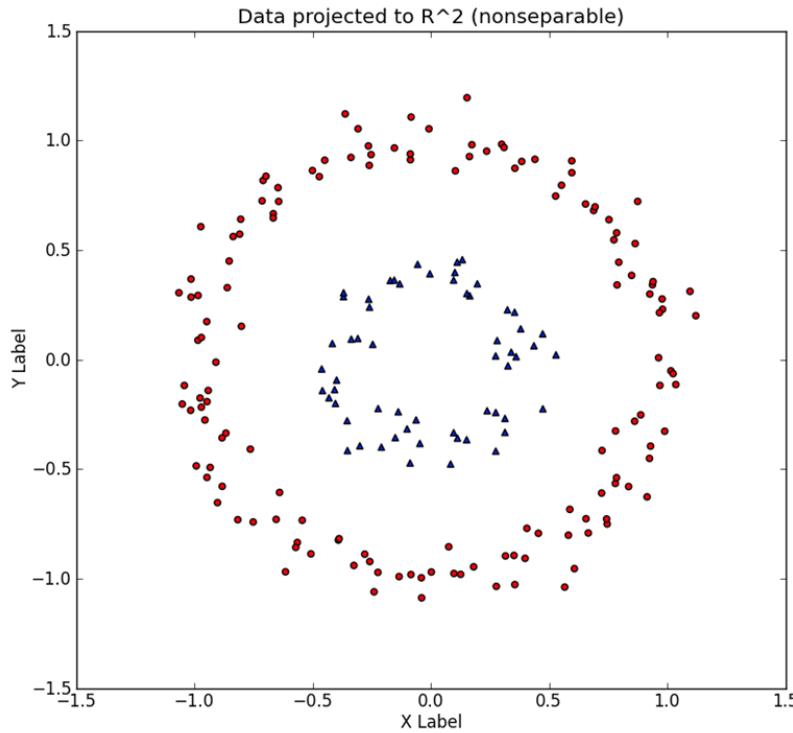
Support Vector Machines

The vector points that the margin lines touch are known as Support Vectors.



Support Vector Machines

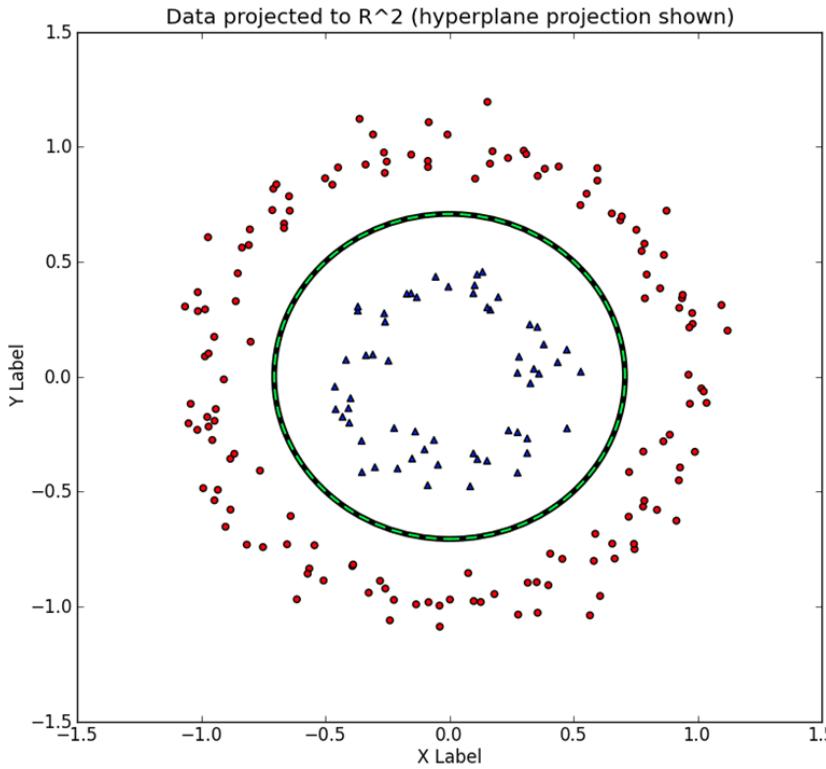
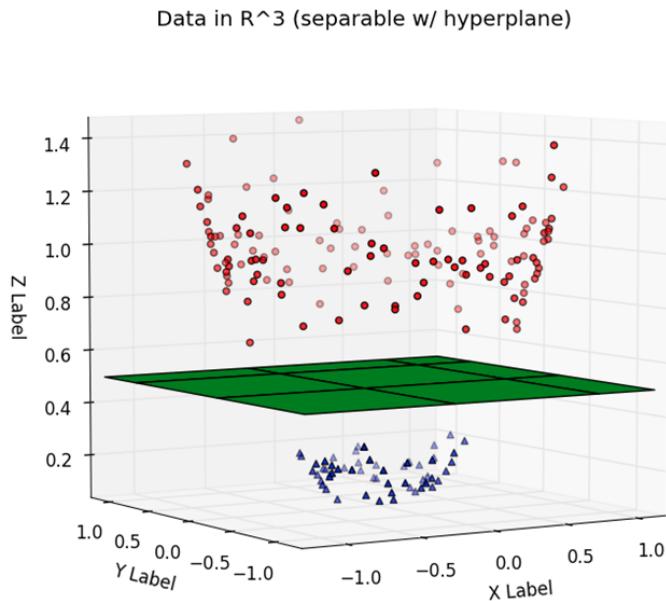
We can expand this idea to non-linearly separable data through the “kernel trick”.





Support Vector Machines

Check out [YouTube](#) for nice 3D Visualization videos explaining this idea. Refer to reading for math behind this.



Unsupervised Learning

Unsupervised Learning

- If Data Science is a mix between an art and a mathematical science, unsupervised learning is where we get to dive deeper into the art.

Unsupervised Learning

- Supervised Learning
 - Using historical **labeled** data, predict a label on new data (regression or classification).
- Unsupervised Learning
 - Using **unlabeled** data, discover patterns, clusters, or significant components.

Unsupervised Learning

- Unsupervised Learning:
 - Clustering:
 - Using features, group together data rows into distinct clusters.
 - Dimensionality Reduction:
 - Using features, discover how to combine and reduce into fewer components.

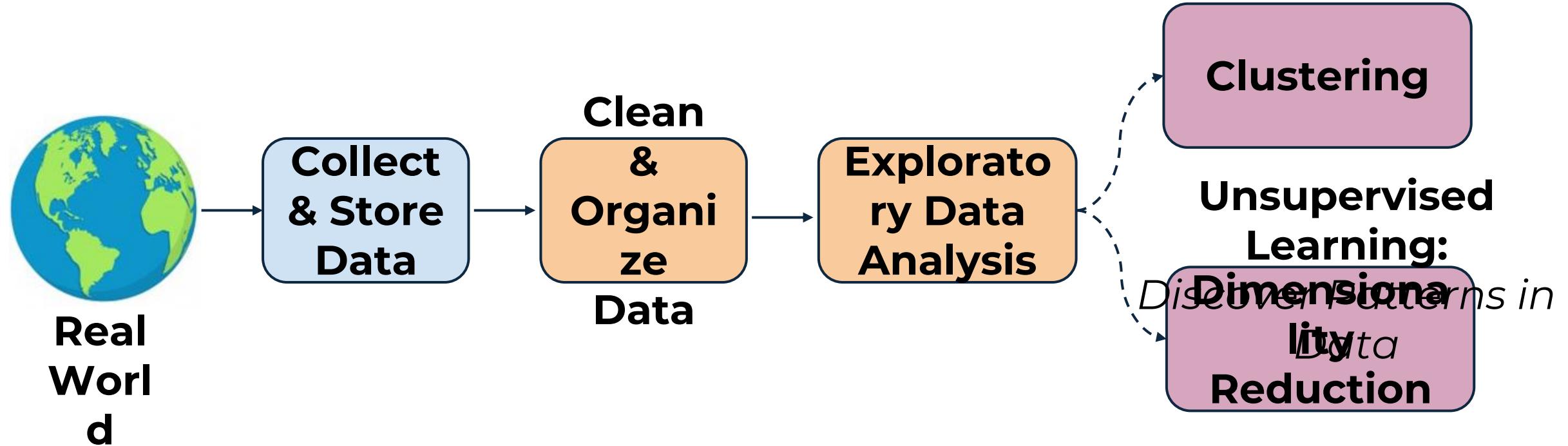
Unsupervised Learning

- Paradigm shift for supervised to unsupervised learning:
 - ***Supervised performance metrics will not apply for unsupervised learning!***
 - How can we compare to a correct label answer, if there was no label to begin with?

Unsupervised Learning

- Instead of metrics like RMSE or Accuracy, we will need to figure out other ways of assessing unsupervised model performance or reasonableness.
- Even our understanding of what “performance” actually means will need to change with unsupervised learning!

ML Pathway

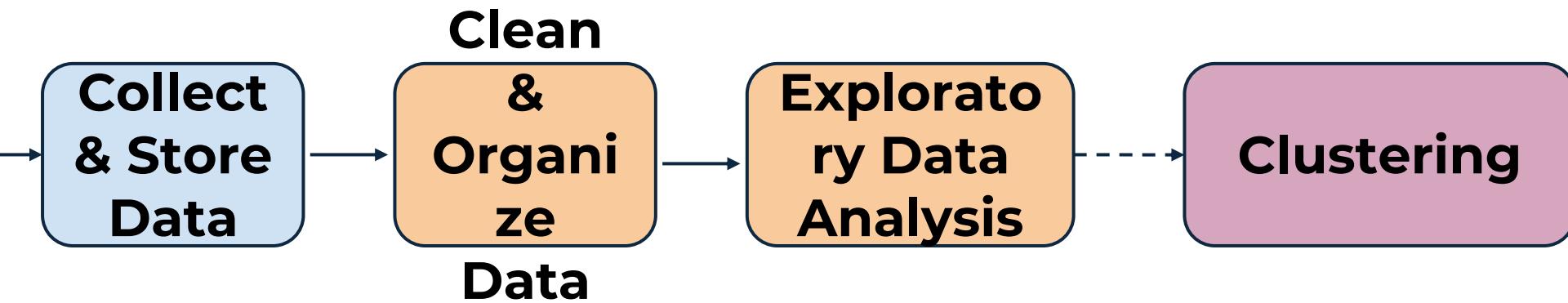


ML Pathway



Real
Worl

d



Clustering: If we have unlabeled data, can we attempt to cluster or group similar data points together to “discover” possible labels for clusters?

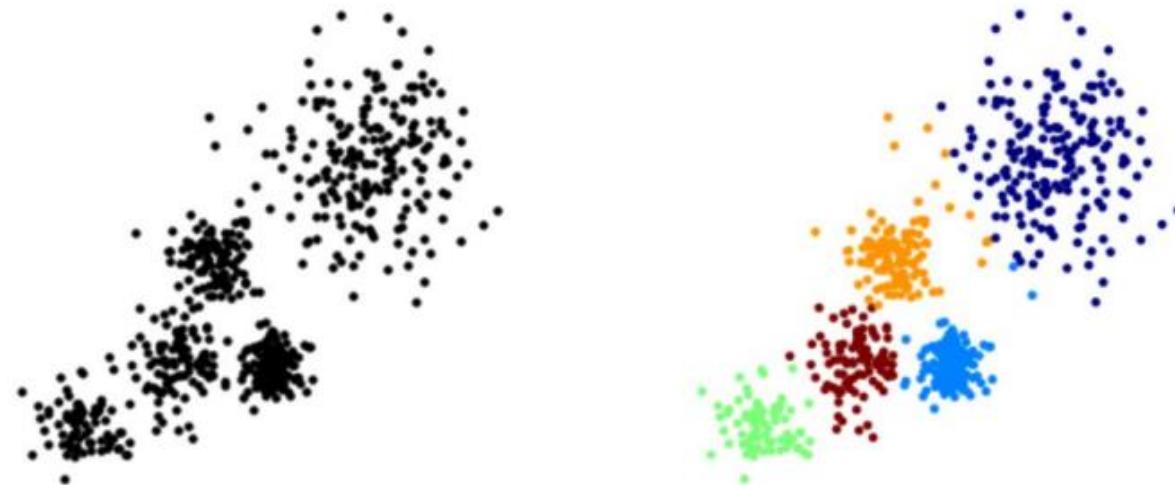
Introduction to K Means Clustering

K Means Clustering is an unsupervised learning algorithm that will attempt to group similar clusters together in your data.

So what does a typical clustering problem look like?

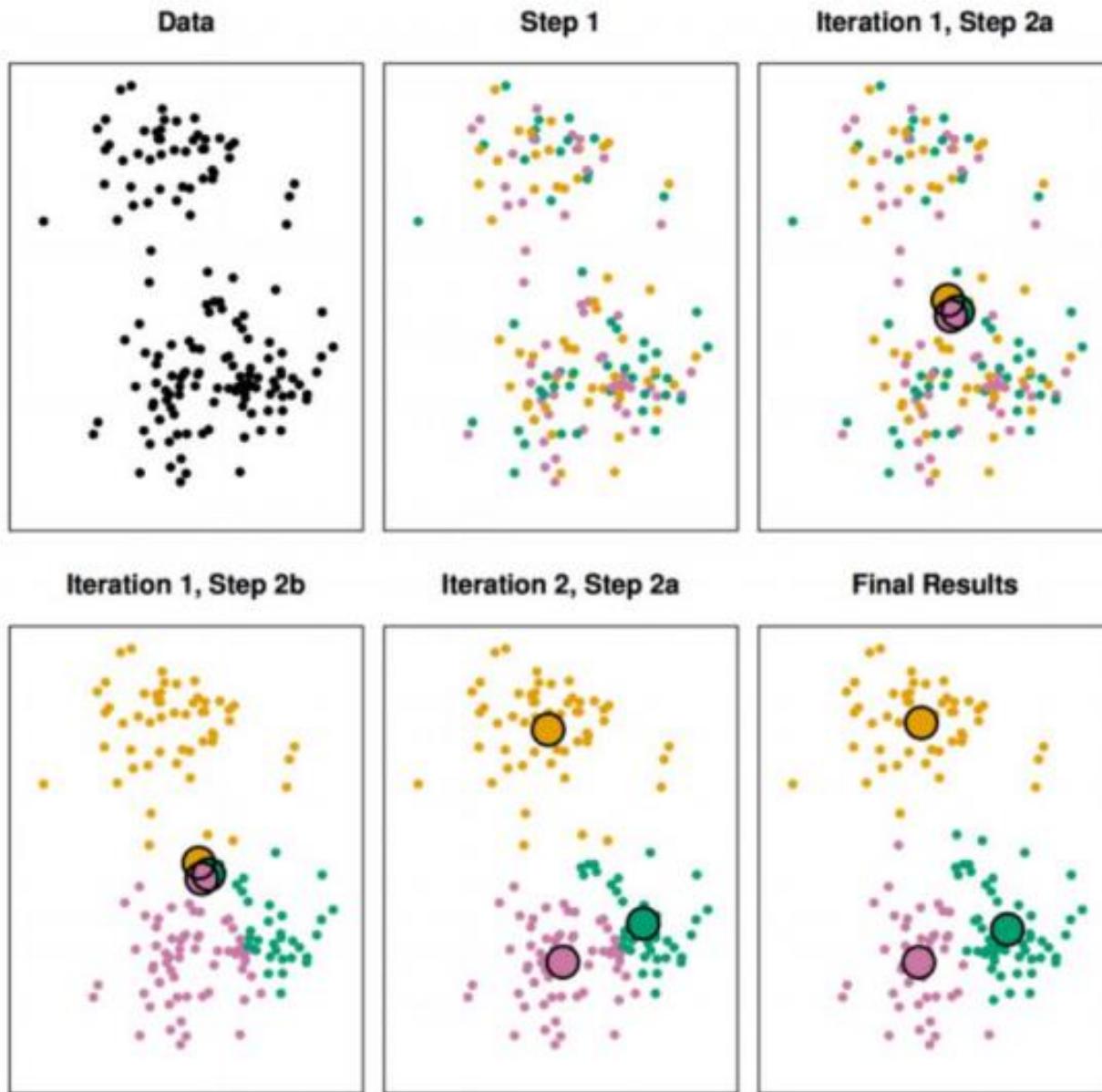
- Cluster Similar Documents
- Cluster Customers based on Features
- Market Segmentation
- Identify similar physical groups

- The overall goal is to divide data into distinct groups such that observations within each group are similar



The K Means Algorithm

- Choose a number of Clusters “K”
- Randomly assign each point to a cluster
- Until clusters stop changing, repeat the following:
 - For each cluster, compute the cluster centroid by taking the mean vector of points in the cluster
 - Assign each data point to the cluster for which the centroid is the closest



- There is no easy answer for choosing a “best” K value
- One way is the elbow method

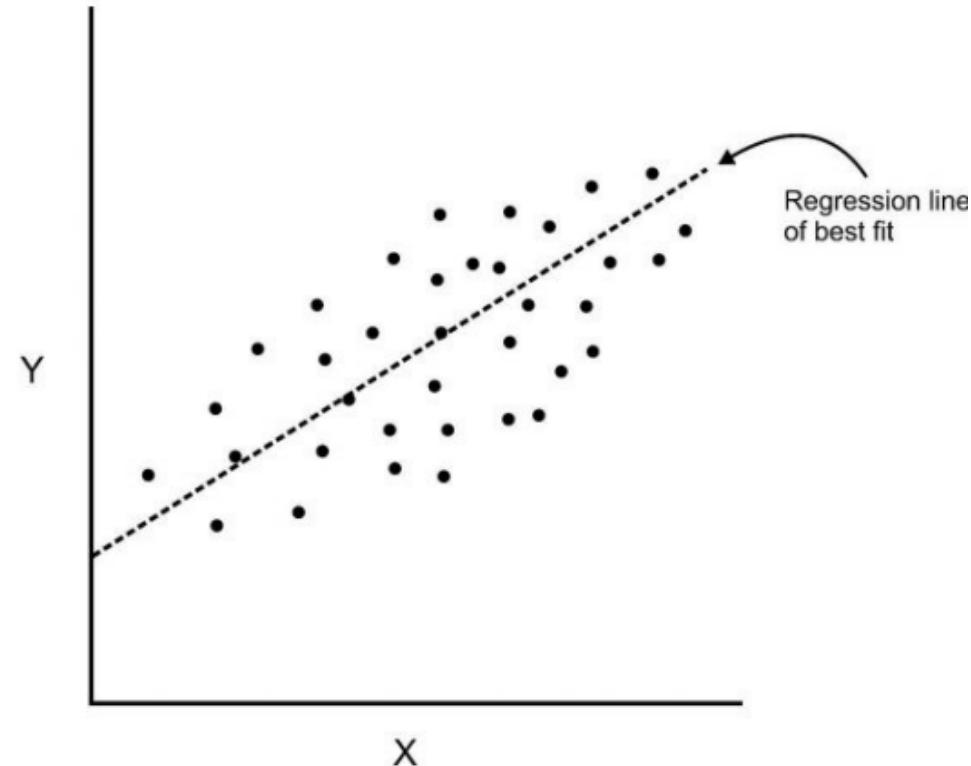
First of all, compute the sum of squared error (SSE) for some values of k (for example 2, 4, 6, 8, etc.).

The SSE is defined as the sum of the squared distance between each member of the cluster and its centroid.

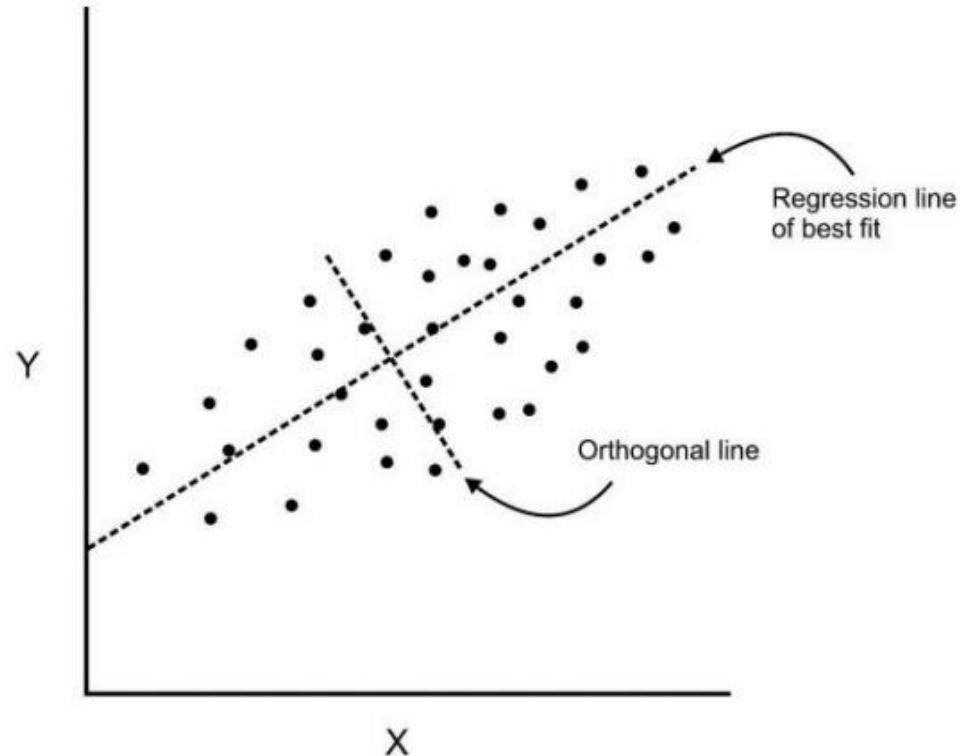
Principal Component Analysis

- Where regression determines a line of best fit to a data set, factor analysis determines several orthogonal lines of best fit to the data set.
- Orthogonal means “at right angles”.
 - Actually the lines are perpendicular to each other in n-dimensional space.
- n-Dimensional Space is the variable sample space.
 - There are as many dimensions as there are variables, so in a data set with 4 variables the sample space is 4-dimensional.

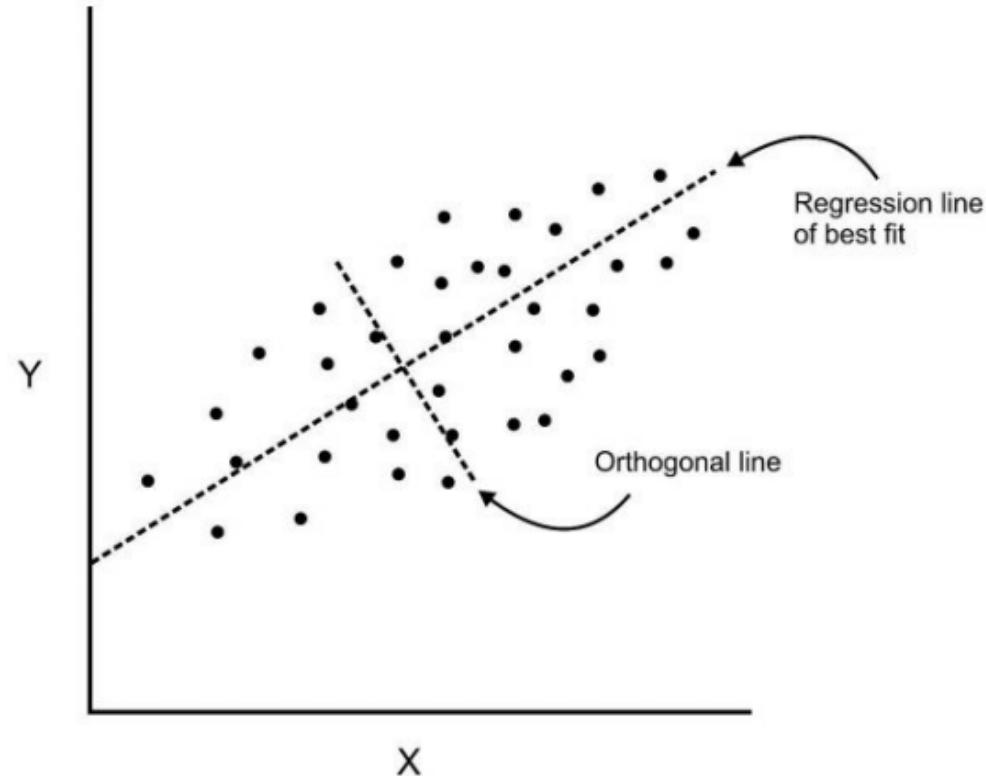
- Here we have some data plotted along two features, x and y.



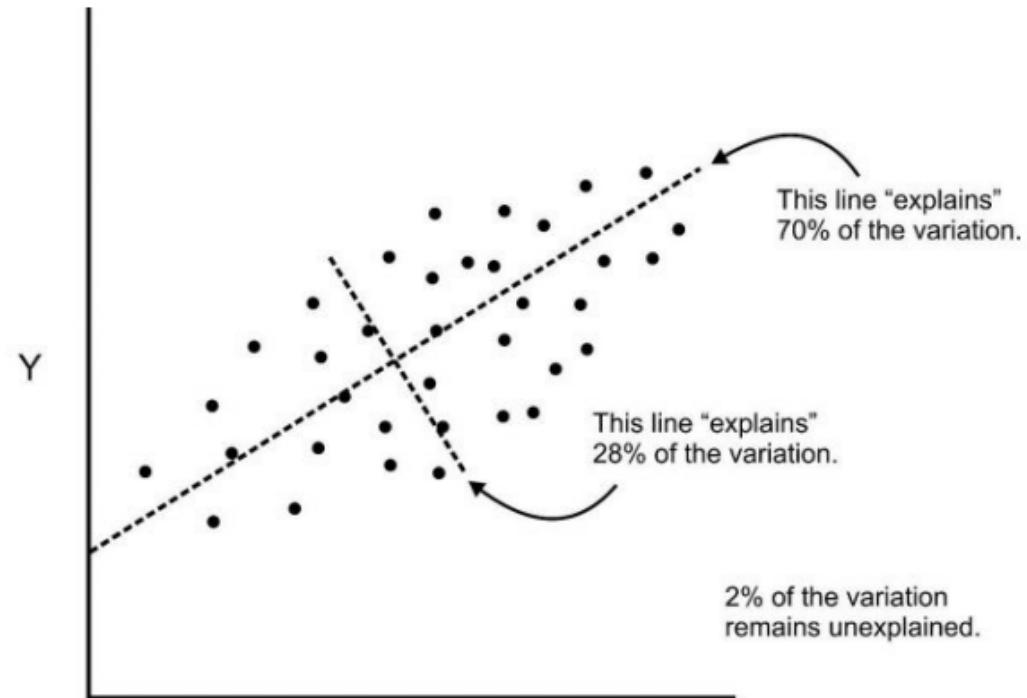
- We can add an orthogonal line.
- Now we can begin to understand the components!



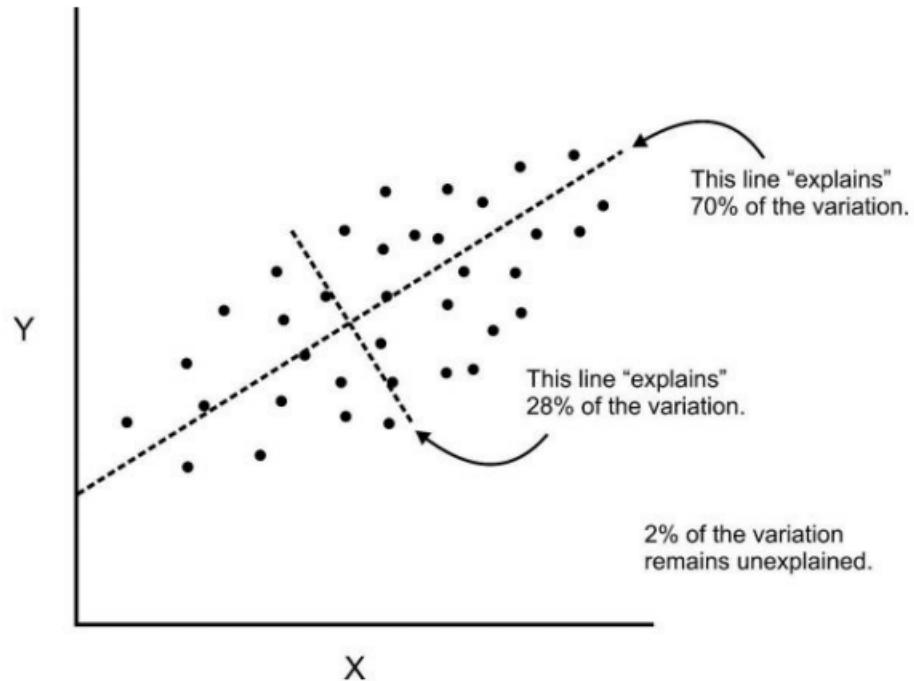
- Components are a linear transformation that chooses a variable system for the data set such that the greatest variance of the data set comes to lie on the first axis



- The second greatest variance on the second axis, and so on ...
- This process allows us to reduce the number of variables used in an analysis.



- Note that components are uncorrelated, since in the sample space they are orthogonal to each other.



- If we use this technique on a data set with a large number of variables, we can compress the amount of explained variation to just a few components.
- The most challenging part of PCA is interpreting the components.

Recommender Systems

The two most common types of recommender systems are **Content-Based** and **Collaborative Filtering (CF)**.

- Collaborative filtering produces recommendations based on the knowledge of users' attitude to items, that is it uses the "wisdom of the crowd" to recommend items.
- Content-based recommender systems focus on the attributes of the items and give you recommendations based on the similarity between them.

- In general, Collaborative filtering (CF) is more commonly used than content-based systems because it usually gives better results and is relatively easy to understand (from an overall implementation perspective).
- The algorithm has the ability to do feature learning on its own, which means that it can start to learn for itself what features to use.

Natural Language Processing

Imagine you work for Google News and you want to group news articles by topic

Or you work for a legal firm and you need to sift through thousands of pages of legal documents to find relevant ones.

This is where NLP can help!

We will want to:

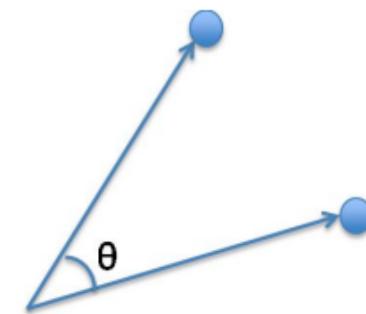
- Compile Documents
- Featurize Them
- Compare their features

Simple Example:

- You have 2 documents:
 - “Blue House”
 - “Red House”
- Featurize based on word count:
 - “Blue House” -> (red,blue,house) -> (0,1,1)
 - “Red House” -> (red,blue,house) -> (1,0,1)

- A document represented as a vector of word counts is called a “Bag of Words”
 - “Blue House” -> (red,blue,house) -> (0,1,1)
 - “Red House” -> (red,blue,house) -> (1,0,1)
- You can use cosine similarity on the vectors made to determine similarity:

$$sim(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$



- We can improve on Bag of Words by adjusting word counts based on their frequency in corpus (the group of all the documents)
- We can use TF-IDF (Term Frequency - Inverse Document Frequency)

- Term Frequency - Importance of the term within that document
 - $TF(d,t) = \text{Number of occurrences of term } t \text{ in document } d$
- Inverse Document Frequency - Importance of the term in the corpus
 - $IDF(t) = \log(D/t)$ where
 - $D = \text{total number of documents}$
 - $t = \text{number of documents with the term}$

- Mathematically, TF-IDF is then expressed:

$$w_{x,y} = tf_{x,y} \times \log \left(\frac{N}{df_x} \right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Natural Language Processing

- Bayes' Theorem
$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}$$
 - **A** and **B** are events
 - **P(A|B)** is probability of event **A** given that **B** is True.
 - **P(B|A)** is probability of event **B** given that **A** is True.
 - **P(A)** is probability of A occurring.
 - **P(B)** is probability of B occurring.

Natural Language Processing

- Bayes' Theorem

$$P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}$$

Natural Language Processing

- There are many variations of Naive Bayes models, including:
 - Multinomial Naive Bayes
 - Gaussian Naive Bayes
 - Bernoulli Naive Bayes
 - Many others!

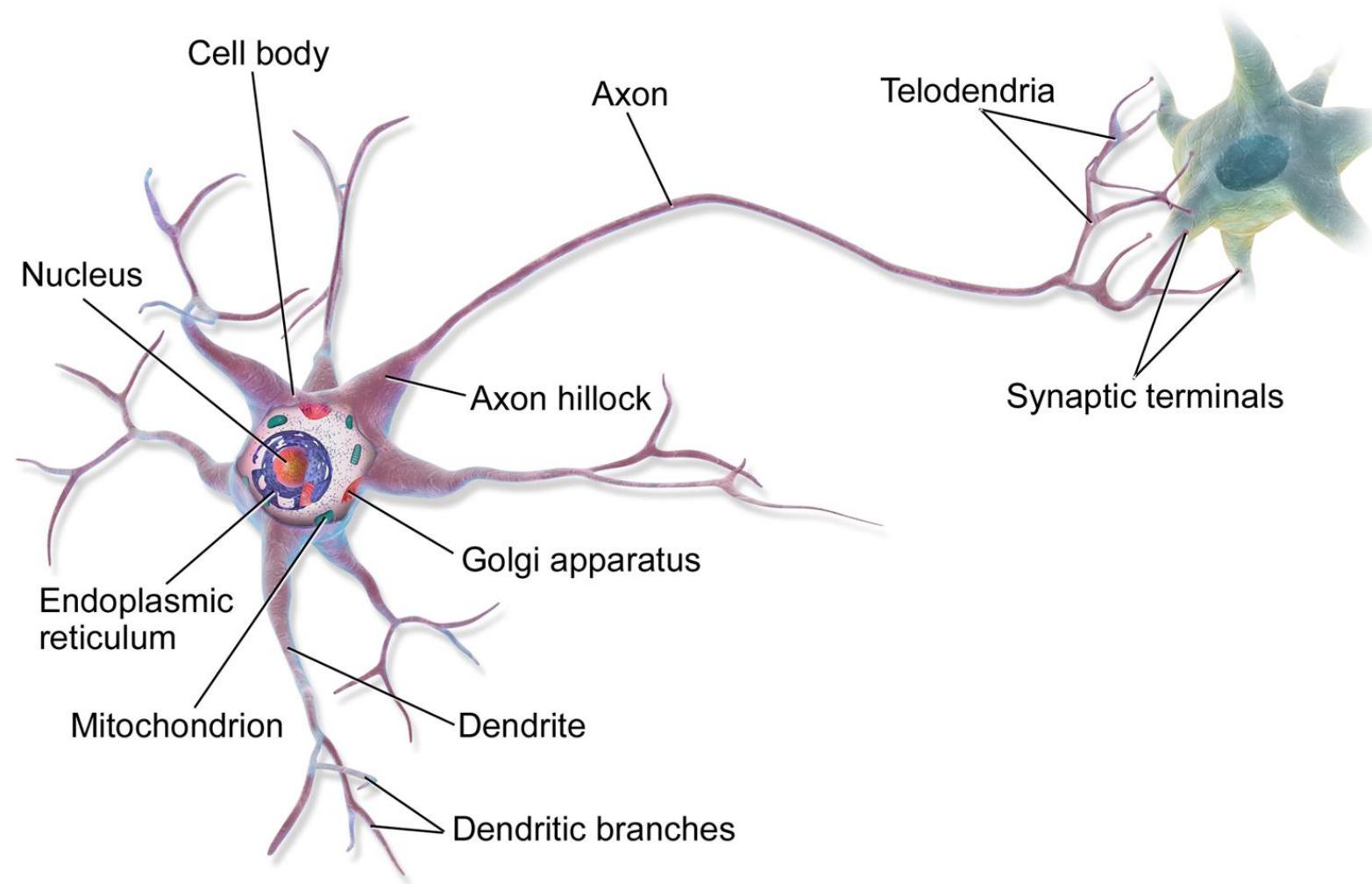
Artificial Neural Networks

Perceptron model

- To begin understanding deep learning, we will build up our model abstractions:
 - Single Biological Neuron
 - Perceptron
 - Multi-layer Perceptron Model
 - Deep Learning Neural Network

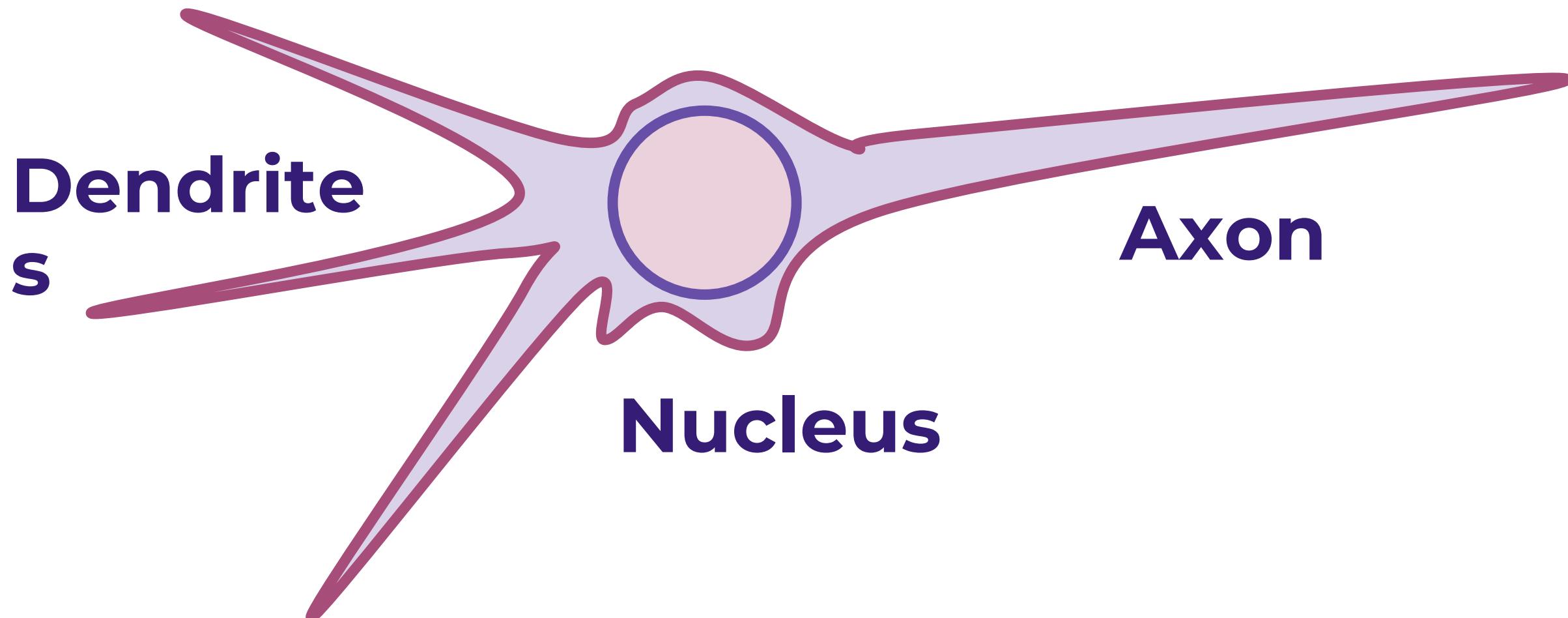
Perceptron model

- Illustration of biological neurons

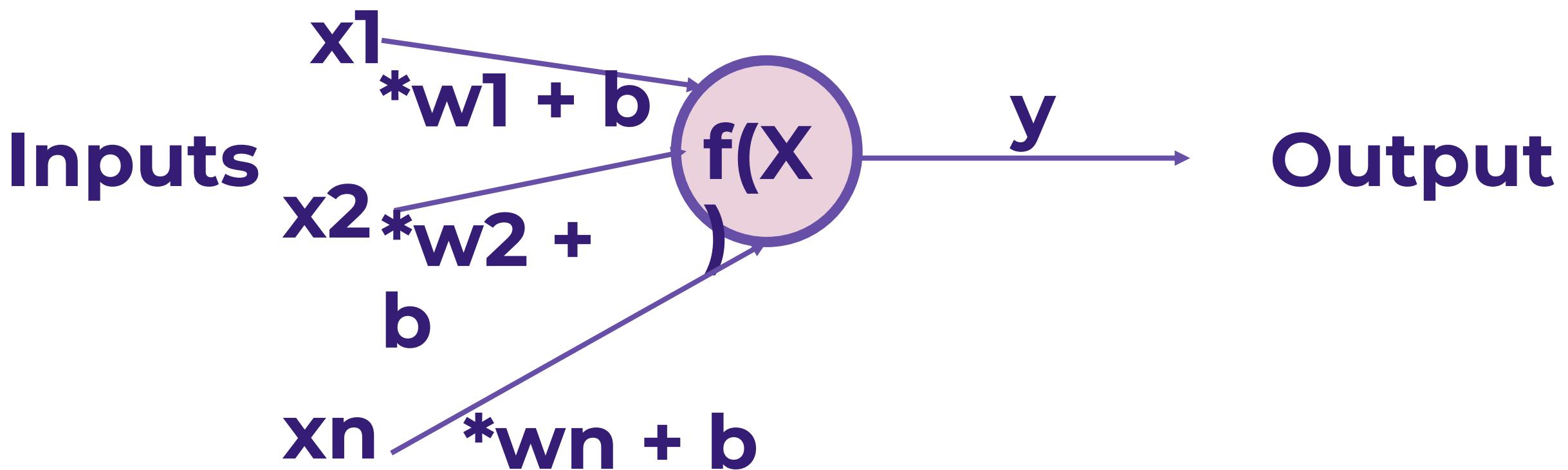


Perceptron model

- Simplified Biological Neuron Model



Perceptron model



Neural Networks

Neural Networks

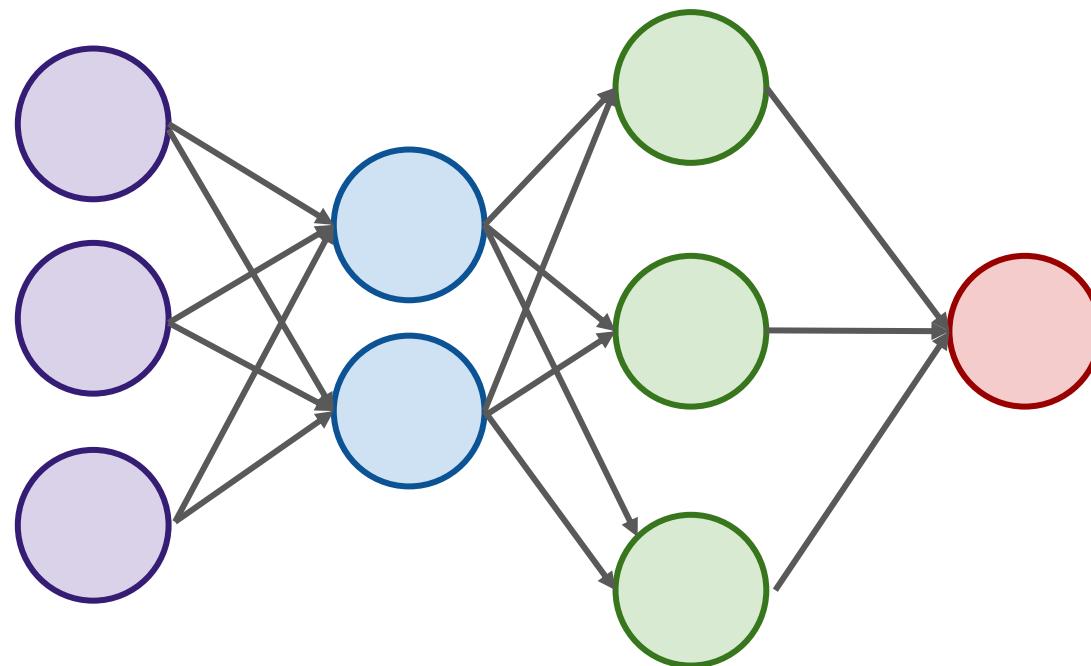
- A single perceptron won't be enough to learn complicated systems.
- Fortunately, we can expand on the idea of a single perceptron, to create a multi-layer perceptron model.

Neural Networks

- A single perceptron won't be enough to learn complicated systems.
- Fortunately, we can expand on the idea of a single perceptron, to create a multi-layer perceptron model.
- We'll also introduce the idea of activation functions.

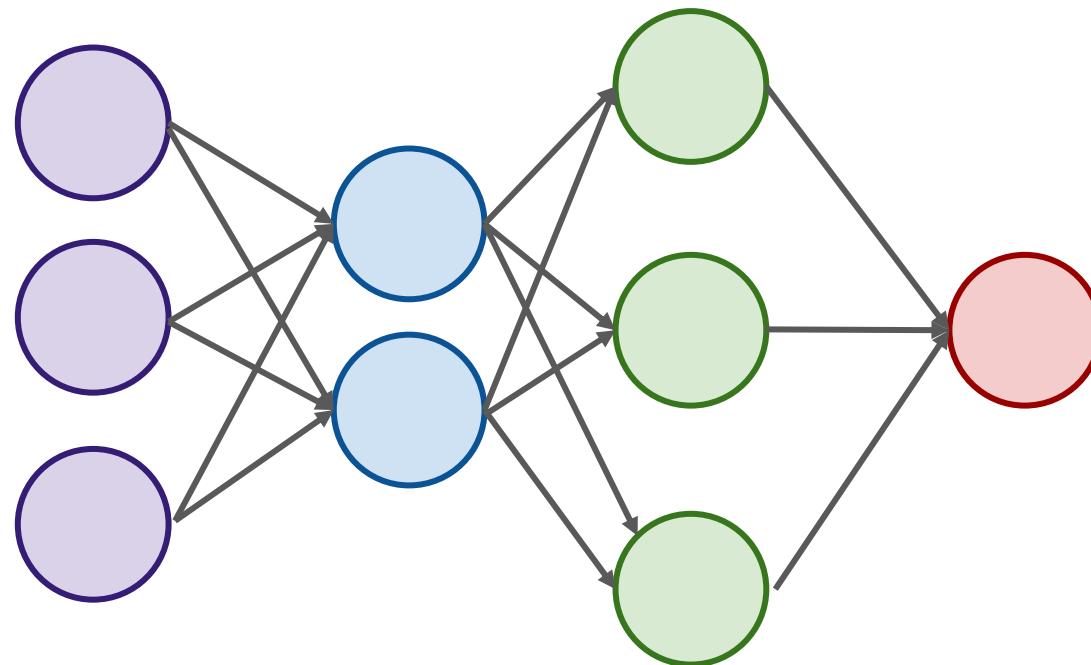
Neural Networks

- To build a network of perceptrons, we can connect layers of perceptrons, using a **multi-layer perceptron model**.



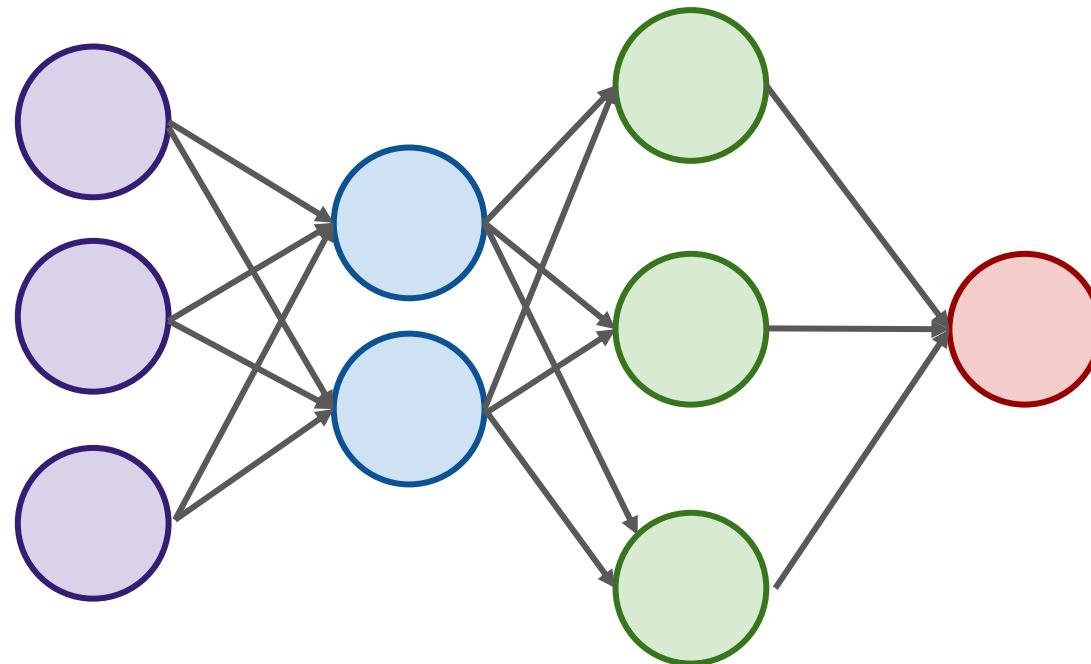
Neural Networks

- The outputs of one perceptron are directly fed into as inputs to another perceptron.



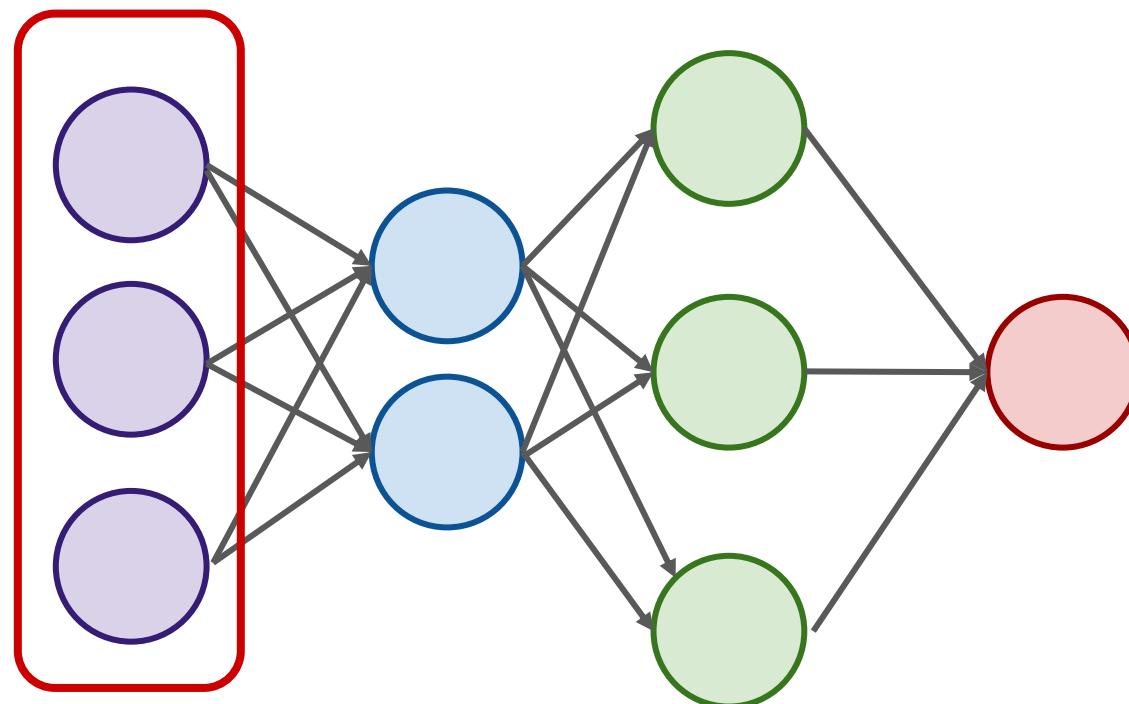
Neural Networks

- This allows the network as a whole to learn about interactions and relationships between features.



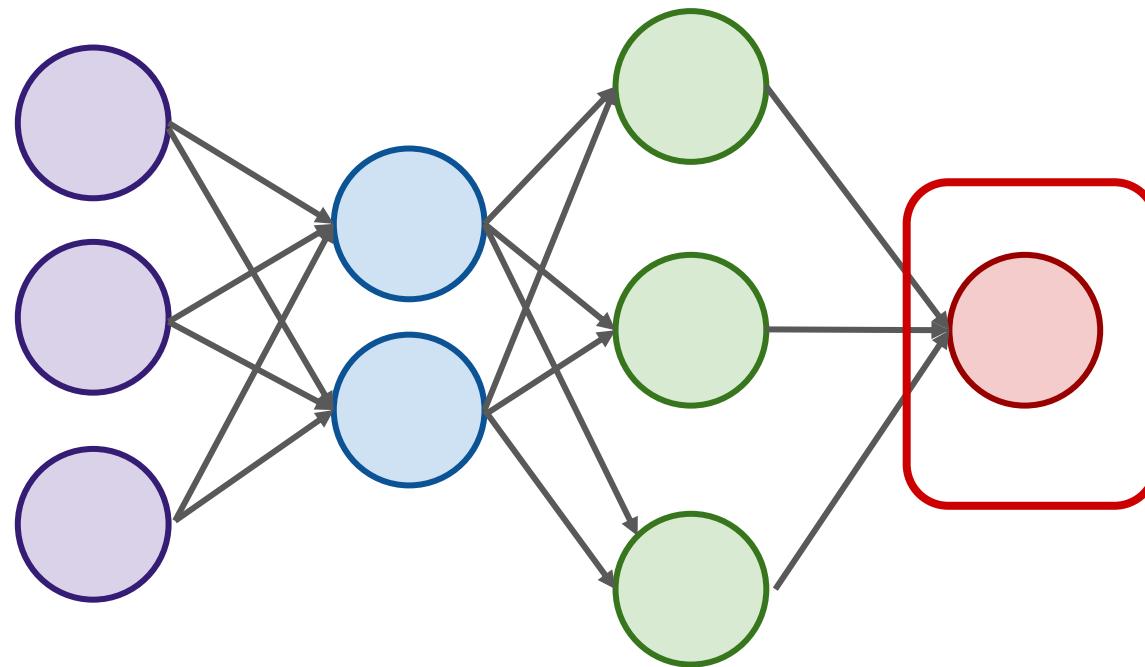
Neural Networks

- The first layer is the **input layer**



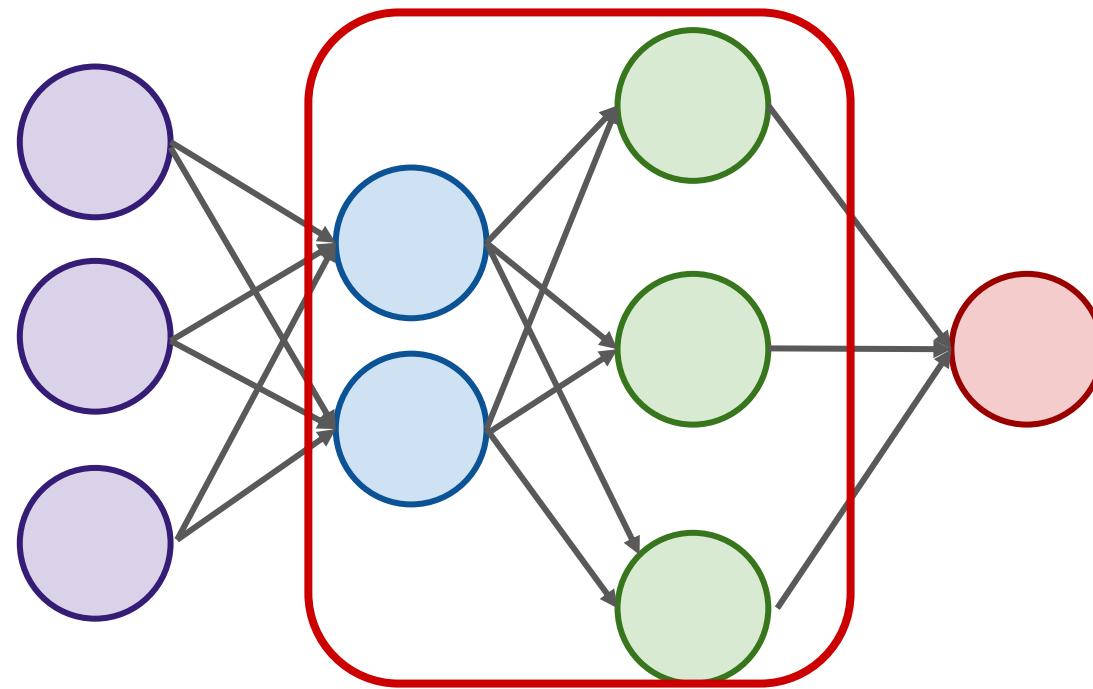
Neural Networks

- The last layer is the **output layer**.
- Note: This last layer can be more than one neuron



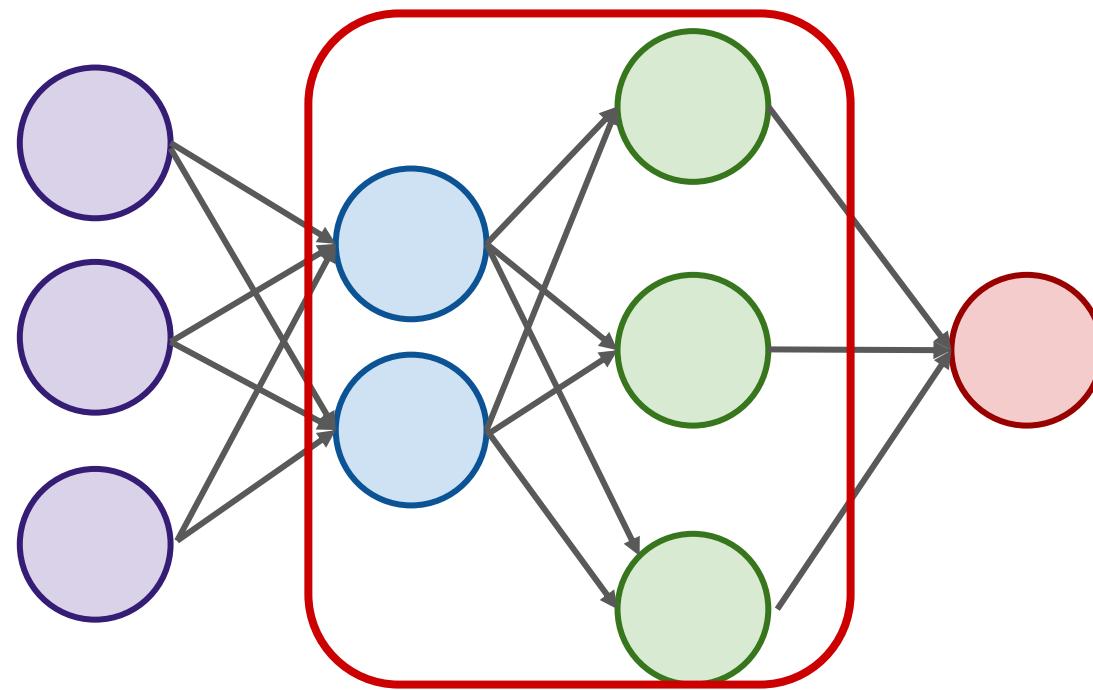
Neural Networks

- Layers in between the input and output layers are the **hidden layers**.



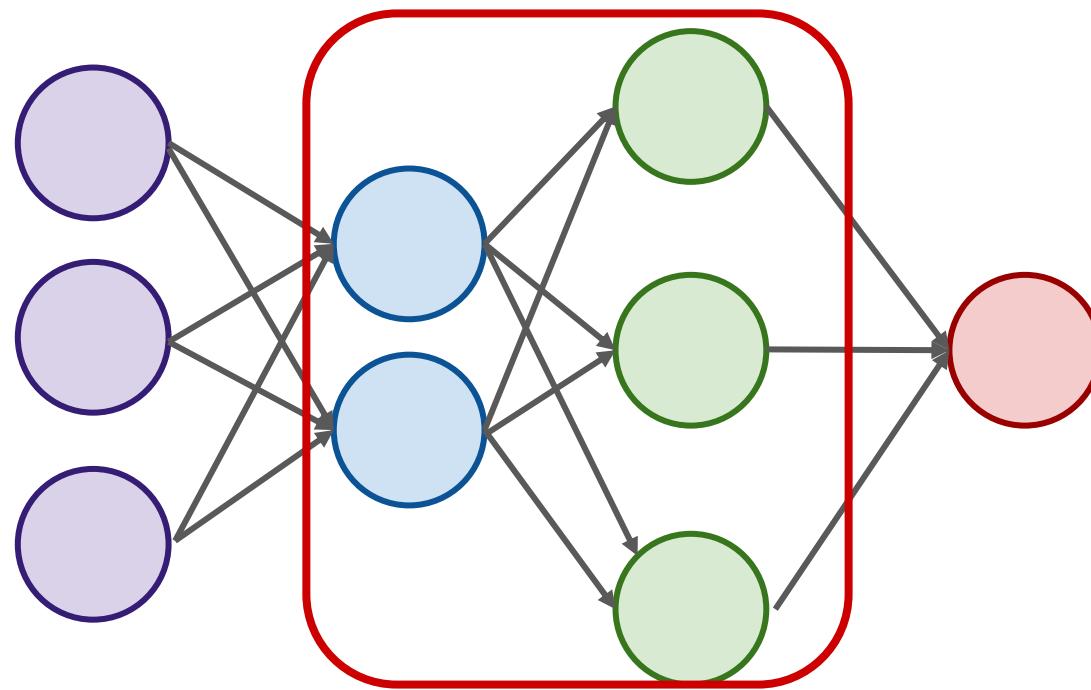
Neural Networks

- Hidden layers are difficult to interpret, due to their high interconnectivity and distance away from known input or output values.



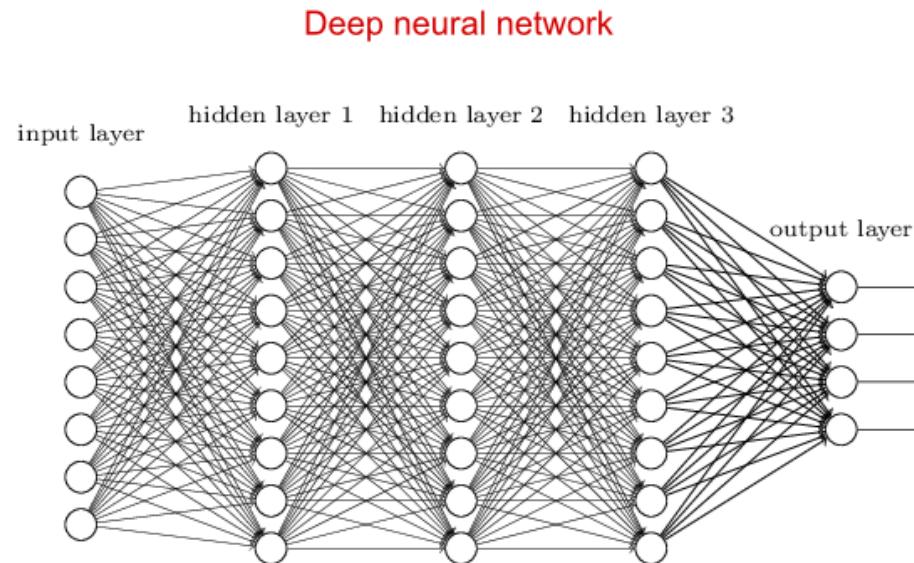
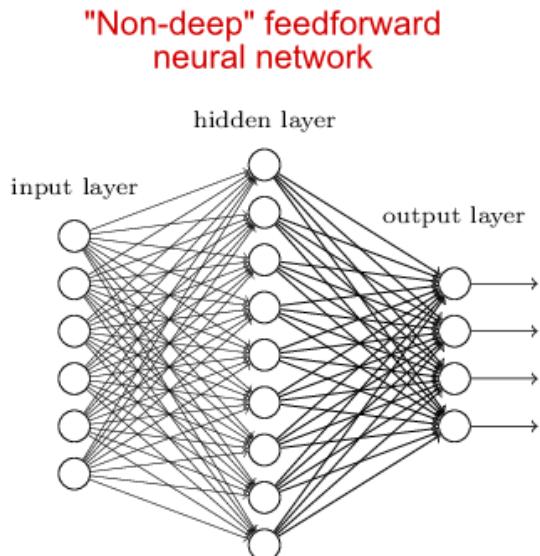
Neural Networks

- Neural Networks become **“deep neural networks”** if they contain 2 or more hidden layers.



Neural Networks

- Neural Networks become **“deep neural networks”** if they contain 2 or more hidden layers.



Neural Networks

- Terminology:
 - Input Layer: First layer that directly accepts real data values
 - Hidden Layer: Any layer between input and output layers
 - Output Layer: The final estimate of the output.

Neural Networks

- What is incredible about the neural network framework is that it can be used to approximate any function.
- Zhou Lu and later on Boris Hanin proved mathematically that Neural Networks can approximate any convex continuous function.

Neural Networks

- Previously in our simple model we saw that the perceptron itself contained a very simple summation function $f(x)$.
- For most use cases however that won't be useful, we'll want to be able to set constraints to our output values, especially in classification tasks.

Neural Networks

- In a classification tasks, it would be useful to have all outputs fall between 0 and 1.
- These values can then present probability assignments for each class.
- In the next lecture, we'll explore how to use **activation functions** to set boundaries to output values from the neuron.

Neural Networks

- In a classification tasks, it would be useful to have all outputs fall between 0 and 1.
- These values can then present probability assignments for each class.
- In the next lecture, we'll explore how to use **activation functions** to set boundaries to output values from the neuron.

Activation Functions

Neural Networks

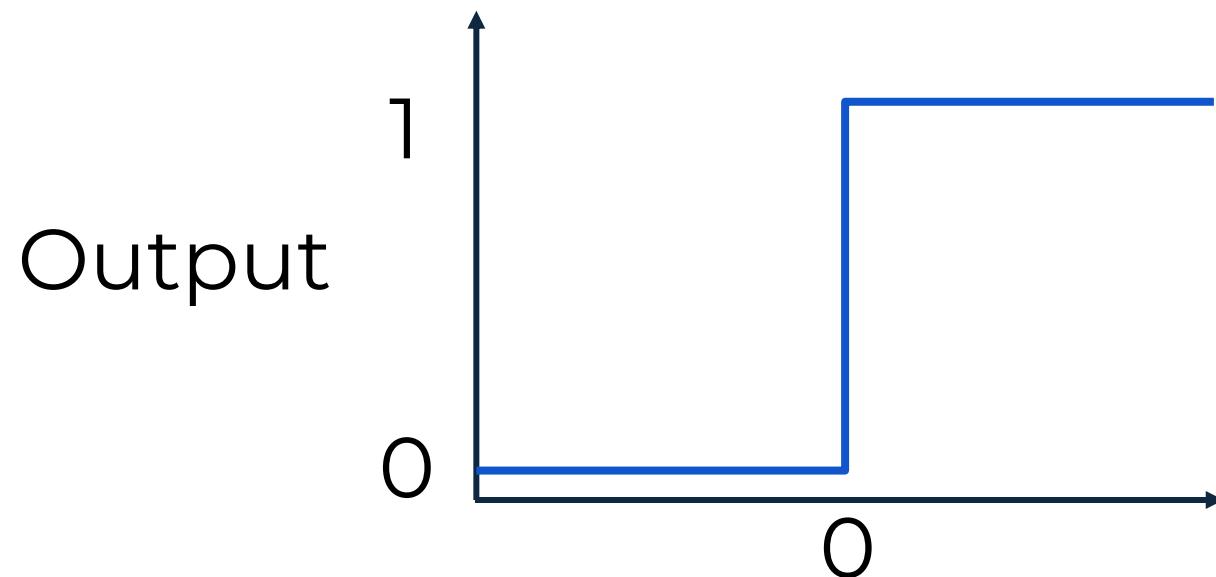
- Recall that inputs \mathbf{x} have a weight \mathbf{w} and a bias term \mathbf{b} attached to them in the perceptron model.
- Which means we have
 - $\mathbf{x}^* \mathbf{w} + \mathbf{b}$

Neural Networks

- For example if $b = -10$
 - $x^*w + b$
 - Then the effects of x^*w won't really start to overcome the bias until their product surpasses 10.
 - After that, then the effect is solely based on the value of w .
 - Thus the term “bias”

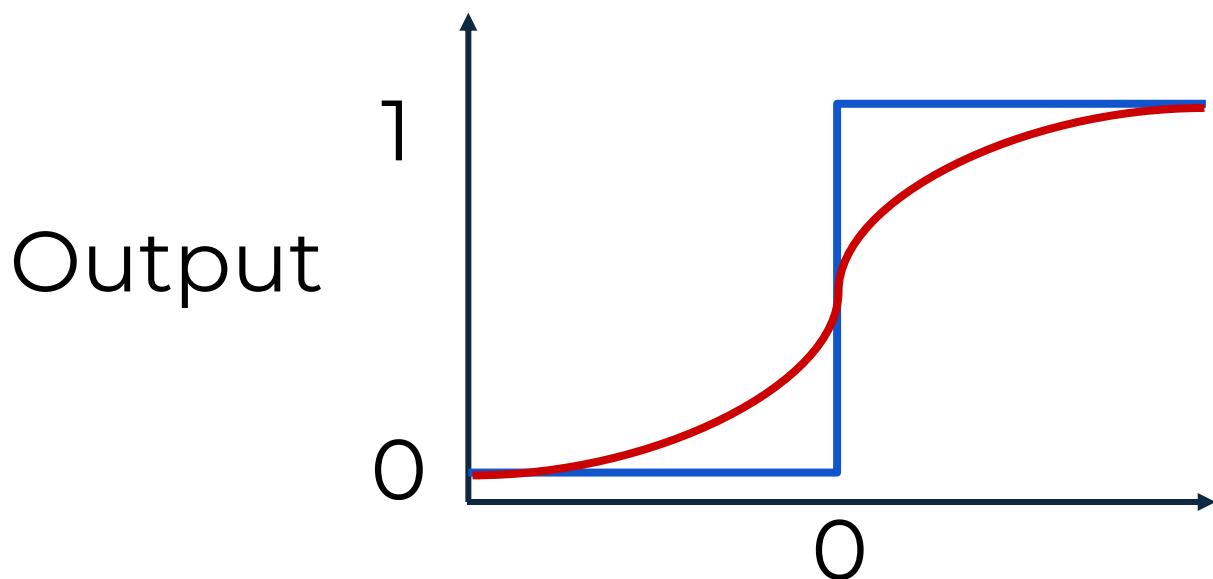
Deep Learning

- The most simple networks rely on a basic **step function** that outputs 0 or 1.



$$z = wx + b$$

Deep Learning



$$f(z) = \frac{1}{1 + e^{(-z)}}$$

Multi-Class Activation Functions

Deep Learning

- Notice all these activation functions make sense for a single output, either a continuous label or trying to predict a binary classification (either a 0 or 1).
- But what should we do if we have a multi-class situation?

Deep Learning

- There are 2 main types of multi-class situations
 - Non-Exclusive Classes
 - A data point can have multiple classes/categories assigned to it
 - Mutually Exclusive Classes
 - Only one class per data point.

Deep Learning

- Non-Exclusive Classes
 - A data point can have multiple classes/categories assigned to it
 - Photos can have multiple tags (e.g. beach, family, vacation, etc...)

Deep Learning

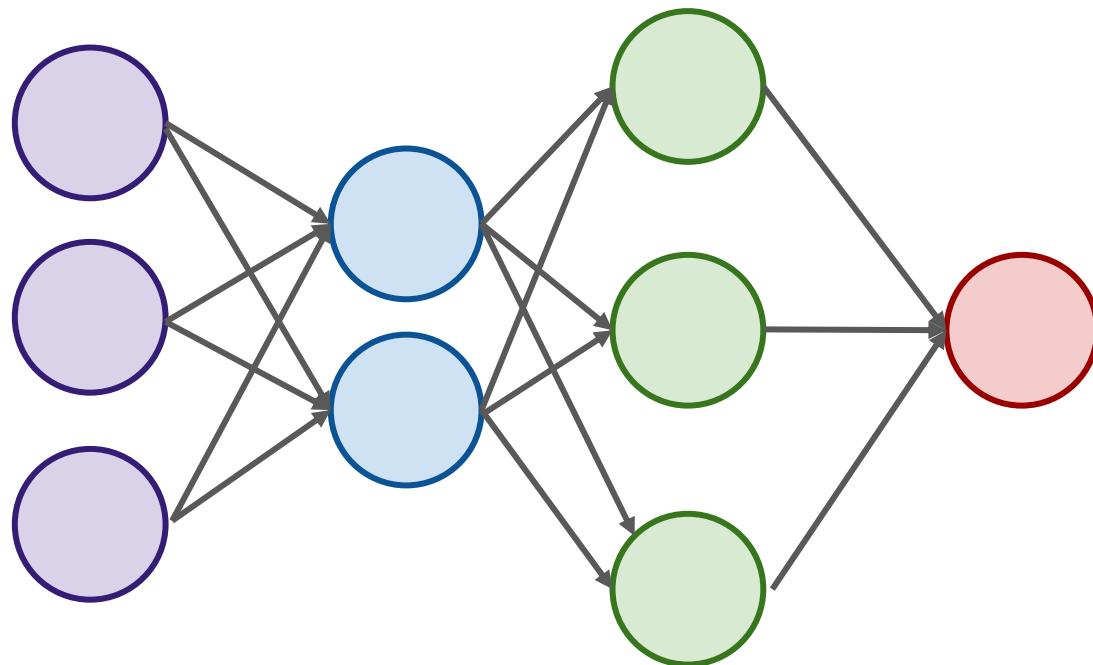
- Mutually Exclusive Classes
 - A data point can only have one class/category assigned to it
 - Photos can be categorized as being in grayscale (black and white) or full color photos. A photo can not be both at the same time.

Deep Learning

- Organizing Multiple Classes
 - The easiest way to organize multiple classes is to simply have 1 output node per class.

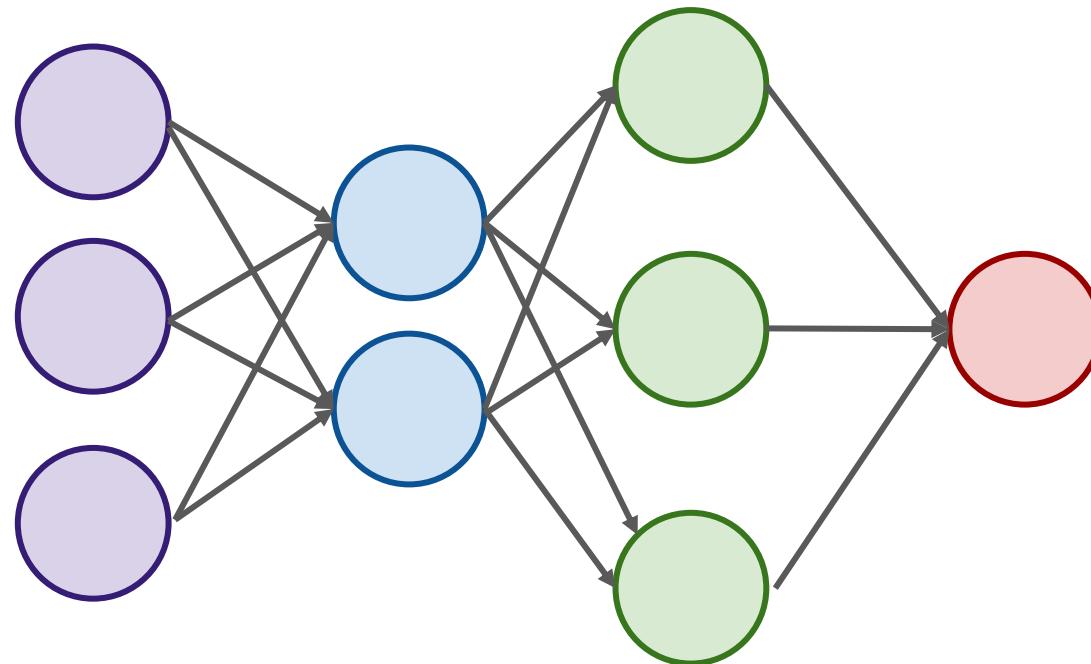
Neural Networks

- Previously we thought of the last output layer as a single node.



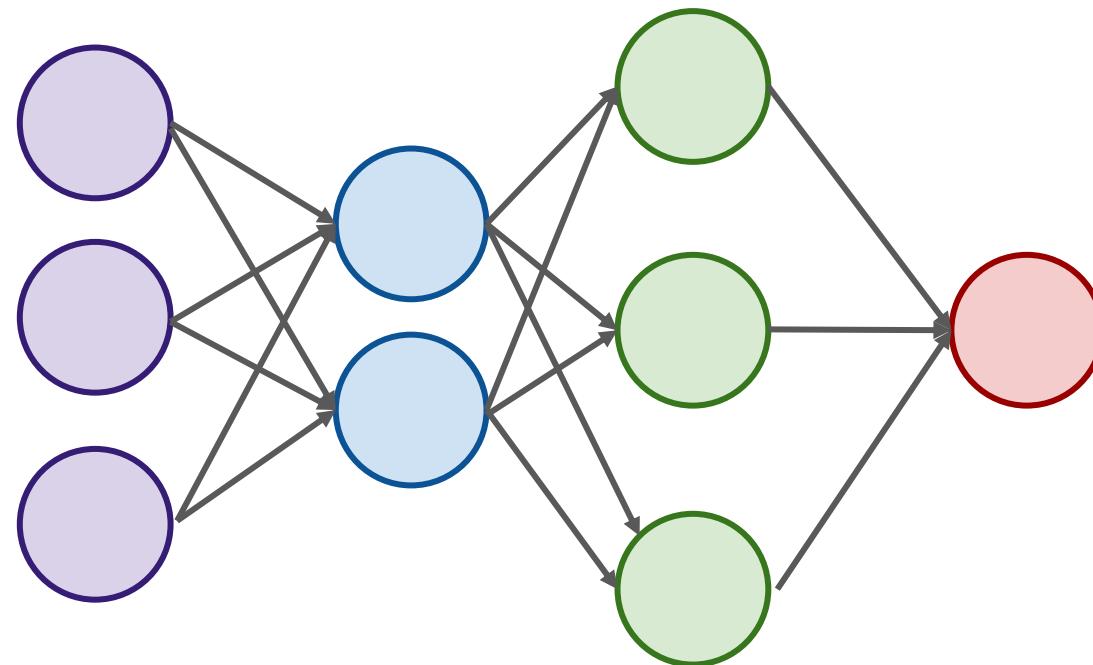
Neural Networks

- This single node could output a continuous regression value or binary classification (0 or 1).



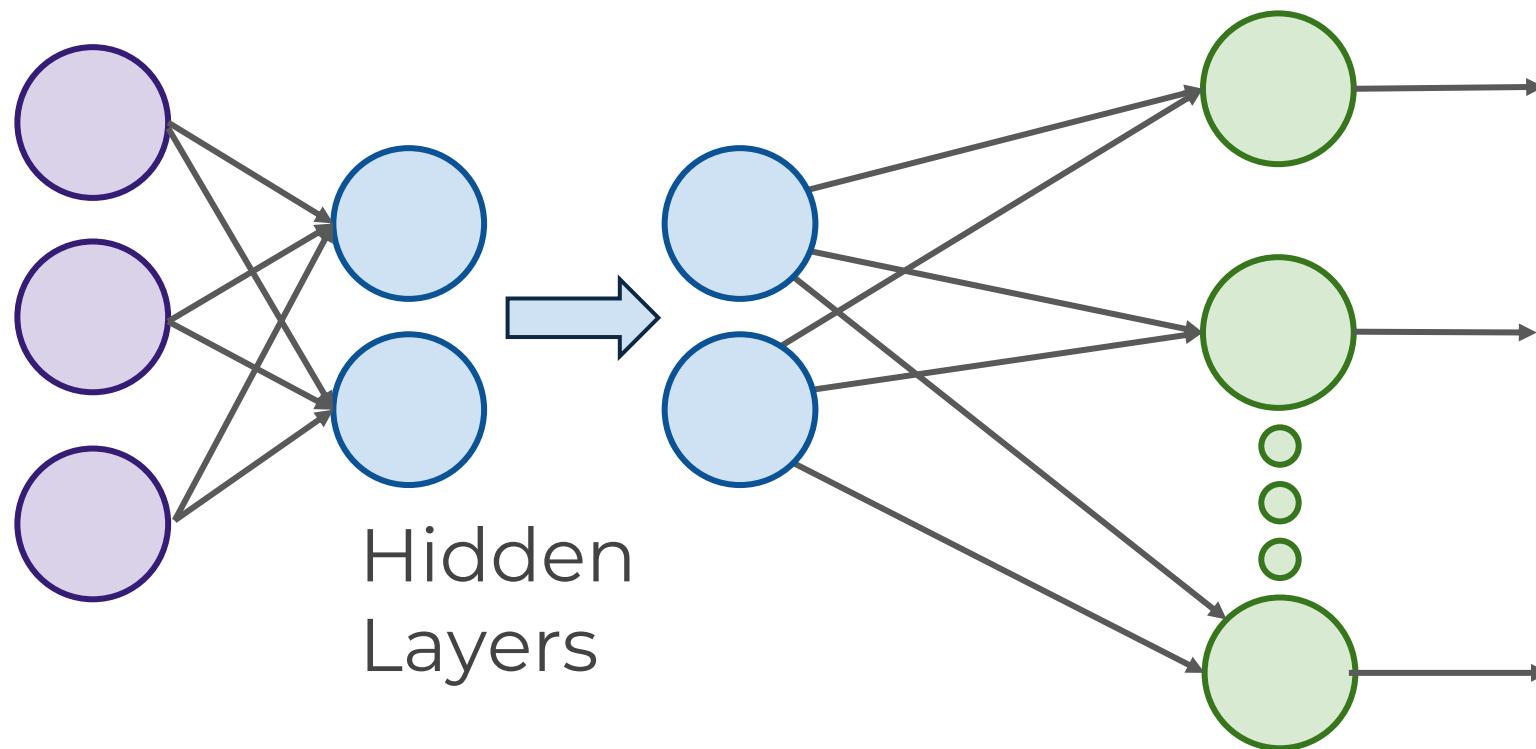
Neural Networks

- Let's expand this output layer to work for the case of multi-classification.



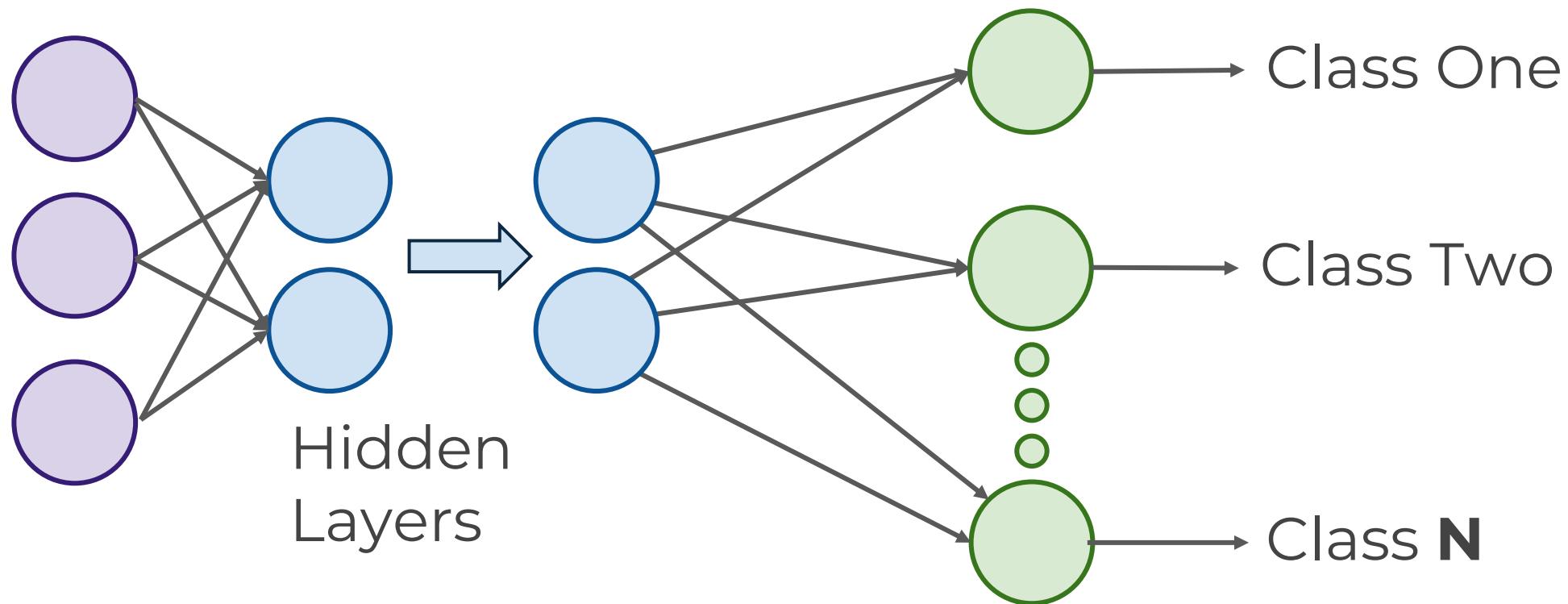
Multiclass Classification

- Organizing for Multiple Classes



Multiclass Classification

- Organizing for Multiple Classes



Deep Learning

- Organizing Multiple Classes
 - This means we will need to organize categories for this output layer.
 - We can't just have categories like "red", "blue", "green", etc...

Deep Learning

- Organizing Multiple Classes
 - Instead we use **one-hot encoding**
 - Let's take a look at what this looks like for mutually exclusive classes.

Deep Learning

- Mutually Exclusive Classes

Data Point 1	RED
Data Point 2	GREEN
Data Point 3	BLUE
...	...
Data Point N	RED

Deep Learning

- Mutually Exclusive Classes

Data Point 1	RED
Data Point 2	GREEN
Data Point 3	BLUE
...	...
Data Point N	RED



	RED	GREEN	BLUE
Data Point 1	1	0	0
Data Point 2	0	1	0
Data Point 3	0	0	1
...

Deep Learning

- Non-Exclusive Classes

Data Point 1	A,B
Data Point 2	A
Data Point 3	C,B
...	...
Data Point N	B



	A	B	C
Data Point 1	1	1	0
Data Point 2	1	0	0
Data Point 3	0	1	1
...

Deep Learning

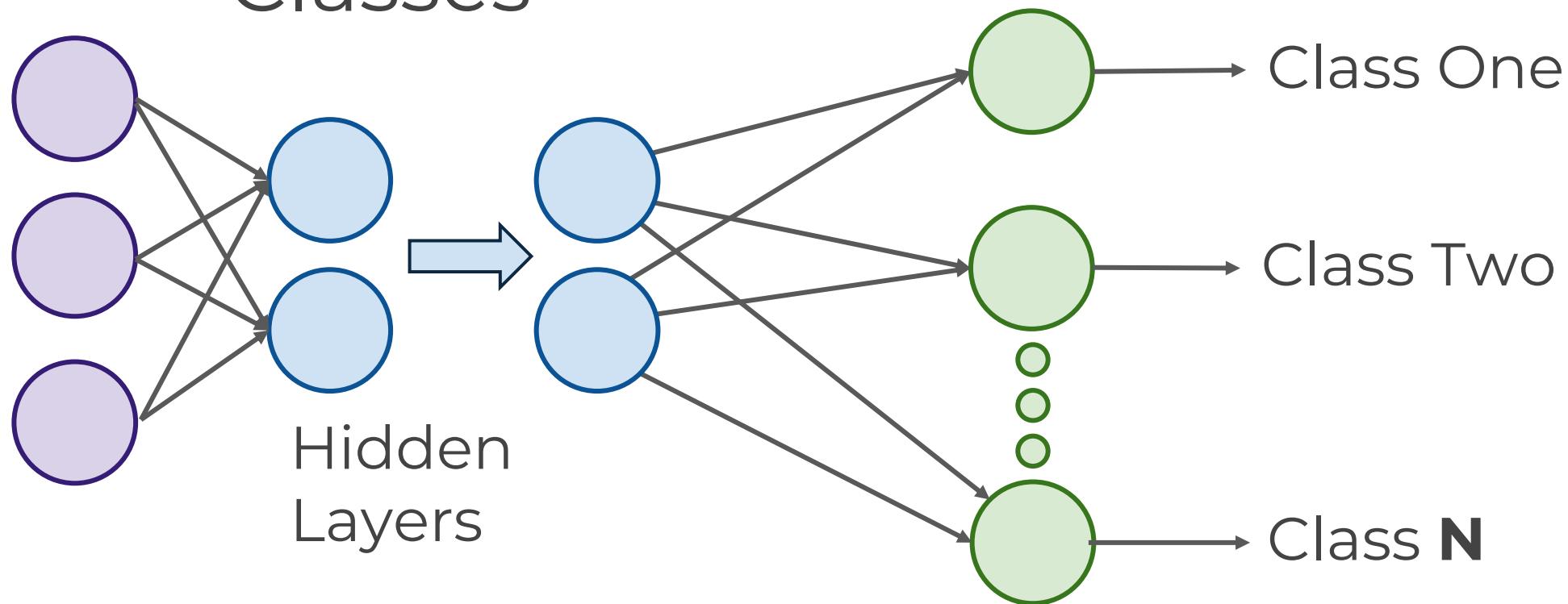
- Now that we have our data correctly organized, we just need to choose the correct classification activation function that the last output layer should have.

Deep Learning

- Non-exclusive
 - Sigmoid function
 - Each neuron will output a value between 0 and 1, indicating the probability of having that class assigned to it.

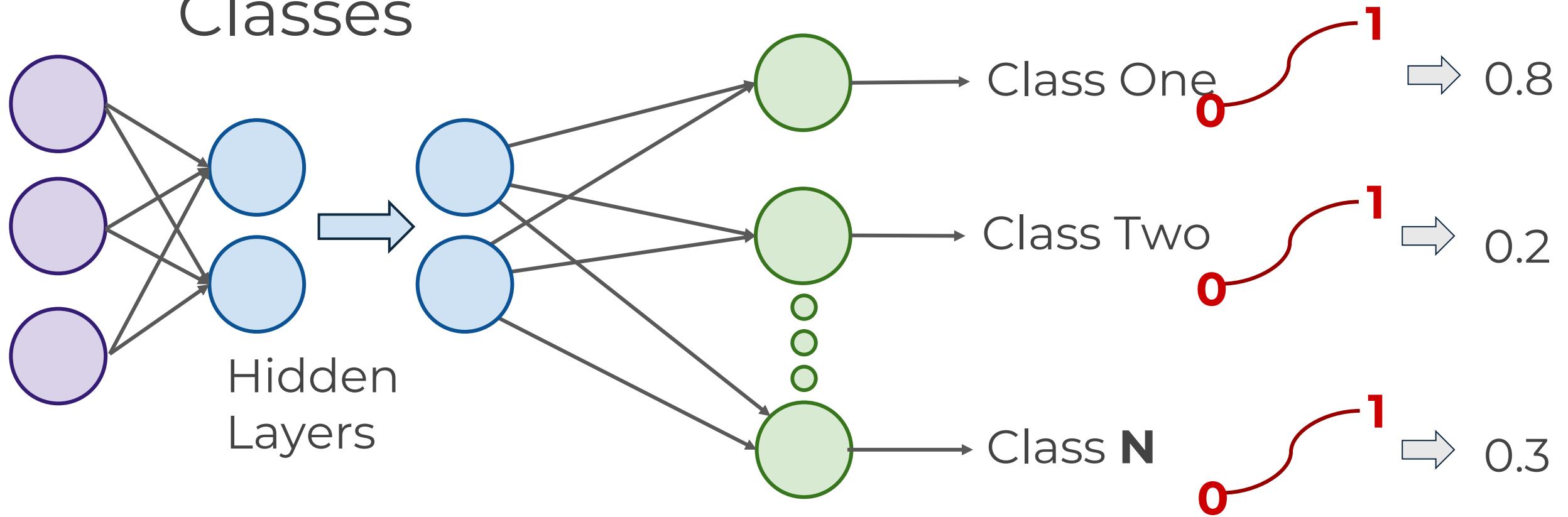
Multiclass Classification

- Sigmoid Function for Non-Exclusive Classes



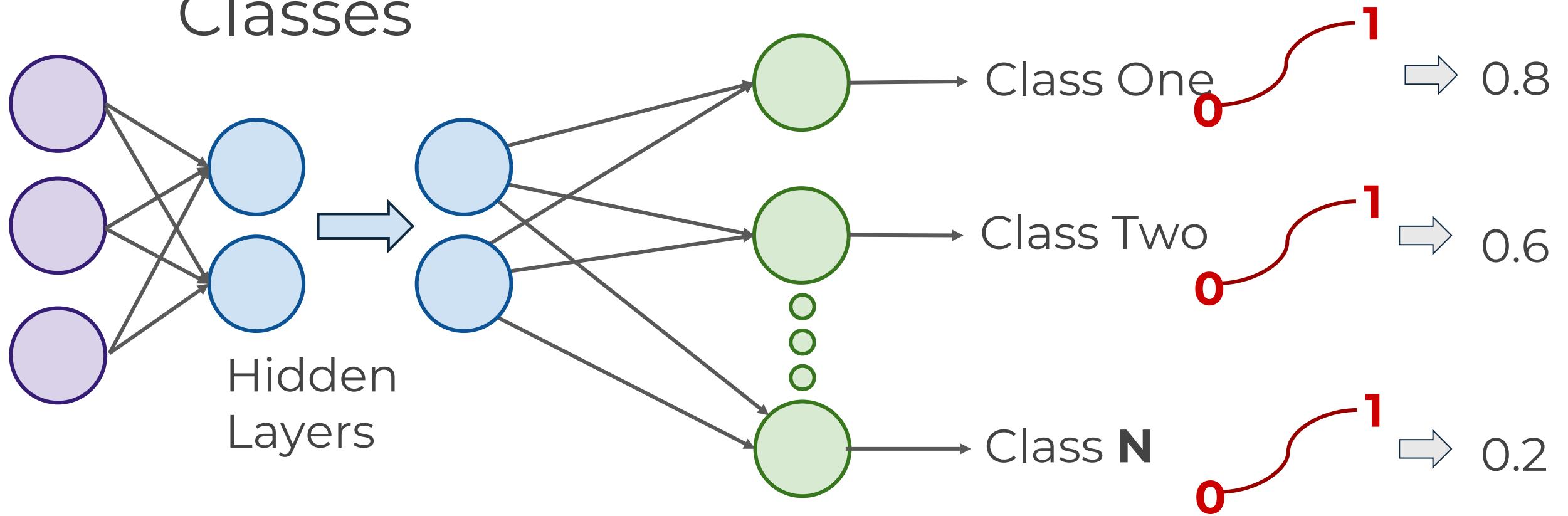
Multiclass Classification

- Sigmoid Function for Non-Exclusive Classes



Multiclass Classification

- Sigmoid Function for Non-Exclusive Classes



Deep Learning

- Non-exclusive
 - Sigmoid function
 - Keep in mind this allows each neuron to output independent of the other classes, allowing for a single data point fed into the function to have multiple classes assigned to it.

Deep Learning

- Mutually Exclusive Classes
 - But what do we do when each data point can only have a single class assigned to it?
 - We can use the **softmax function** for this!

Deep Learning

- Softmax Function

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, \dots, K$$

Deep Learning

- Mutually Exclusive Classes
 - Softmax function calculates the probabilities distribution of the event over **K** different events.
 - This function will calculate the probabilities of each target class over all possible target classes.

Deep Learning

- Mutually Exclusive Classes
 - The range will be 0 to 1, and **the sum of all the probabilities will be equal to one.**
 - The model returns the probabilities of each class and the target class chosen will have the highest probability.

Deep Learning

- Mutually Exclusive Classes
 - The main thing to keep in mind is that if you use softmax for multi-class problems you get this sort of output:
 - [Red , Green , Blue]
 - [0.1 , 0.6 , 0.3]

Deep Learning

- Mutually Exclusive Classes
 - The probabilities for each class all sum up to 1. We choose the highest probability as our assignment.
 - [Red , Green , Blue]
 - [0.1 , 0.6 , 0.3]

Deep Learning

- Review
 - Perceptrons expanded to neural network model
 - Weights and Biases
 - Activation Functions
 - Time to learn about Cost Functions!

Cost Functions and Gradient Descent

Deep Learning

- We now understand that neural networks take in inputs, multiply them by weights, and add biases to them.
- Then this result is passed through an activation function which at the end of all the layers leads to some output.

Deep Learning

- This output \hat{y} is the model's estimation of what it predicts the label to be.
- So after the network creates its prediction, how do we evaluate it?
- And after the evaluation how can we update the network's weights and biases?

Deep Learning

- We need to take the estimated outputs of the network and then compare them to the real values of the label.
- Keep in mind this is using the training data set during the fitting/training of the model.

Deep Learning

- The cost function (often referred to as a loss function) must be an average so it can output a single value.
- We can keep track of our loss/cost during training to monitor network performance.

Deep Learning

- We'll use the following variables:
 - y to represent the true value
 - a to represent neuron's prediction
- In terms of weights and bias:
 - $w^*x + b = z$
 - Pass z into activation function $\sigma(z) = a$

Deep Learning

- One very common cost function is the quadratic cost function:

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

Deep Learning

- We simply calculate the difference between the real values $y(x)$ against our predicted values $a(x)$.

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

Deep Learning

- Note: The notation shown here corresponds to vector inputs and outputs, since we will be dealing with a **batch** of training points and predictions.

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

Deep Learning

- Notice how squaring this does 2 useful things for us, keeps everything positive and **punishes** large errors!

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

Deep Learning

- We can think of the cost function as:

$$C(W, B, S^r, E^r)$$

Deep Learning

- **W** is our neural network's weights, **B** is our neural network's biases, **S^r** is the input of a single training sample, and **E^r** is the desired output of that training sample.

$$C(W, B, S^r, E^r)$$

Deep Learning

- Notice how that information was all encoded in our simplified notation.
- The **a(x)** holds information about weights and biases.

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

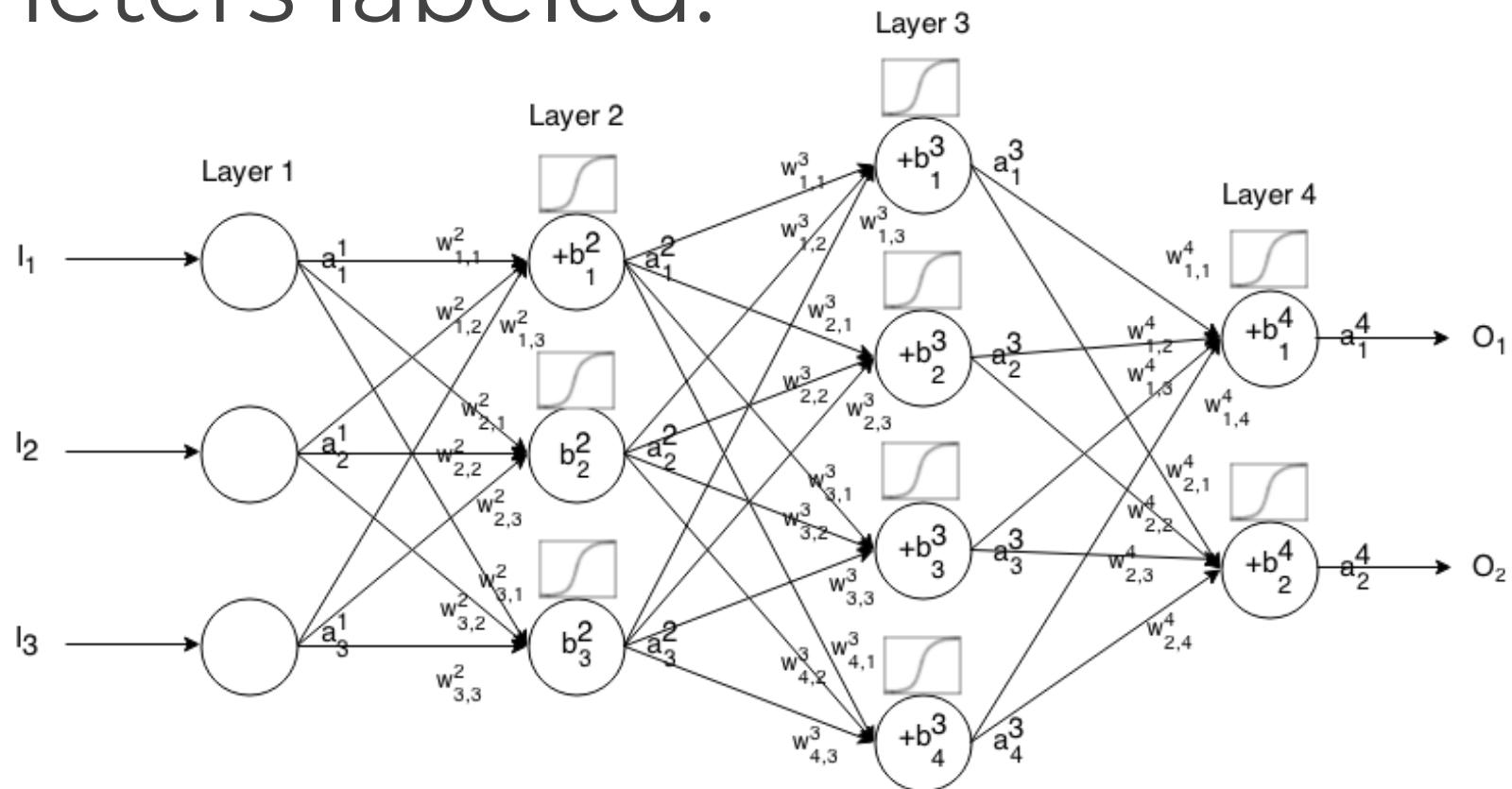
Deep Learning

- This means that if we have a huge network, we can expect **C** to be quite complex, with huge vectors of weights and biases.

$$C(W, B, S^r, E^r)$$

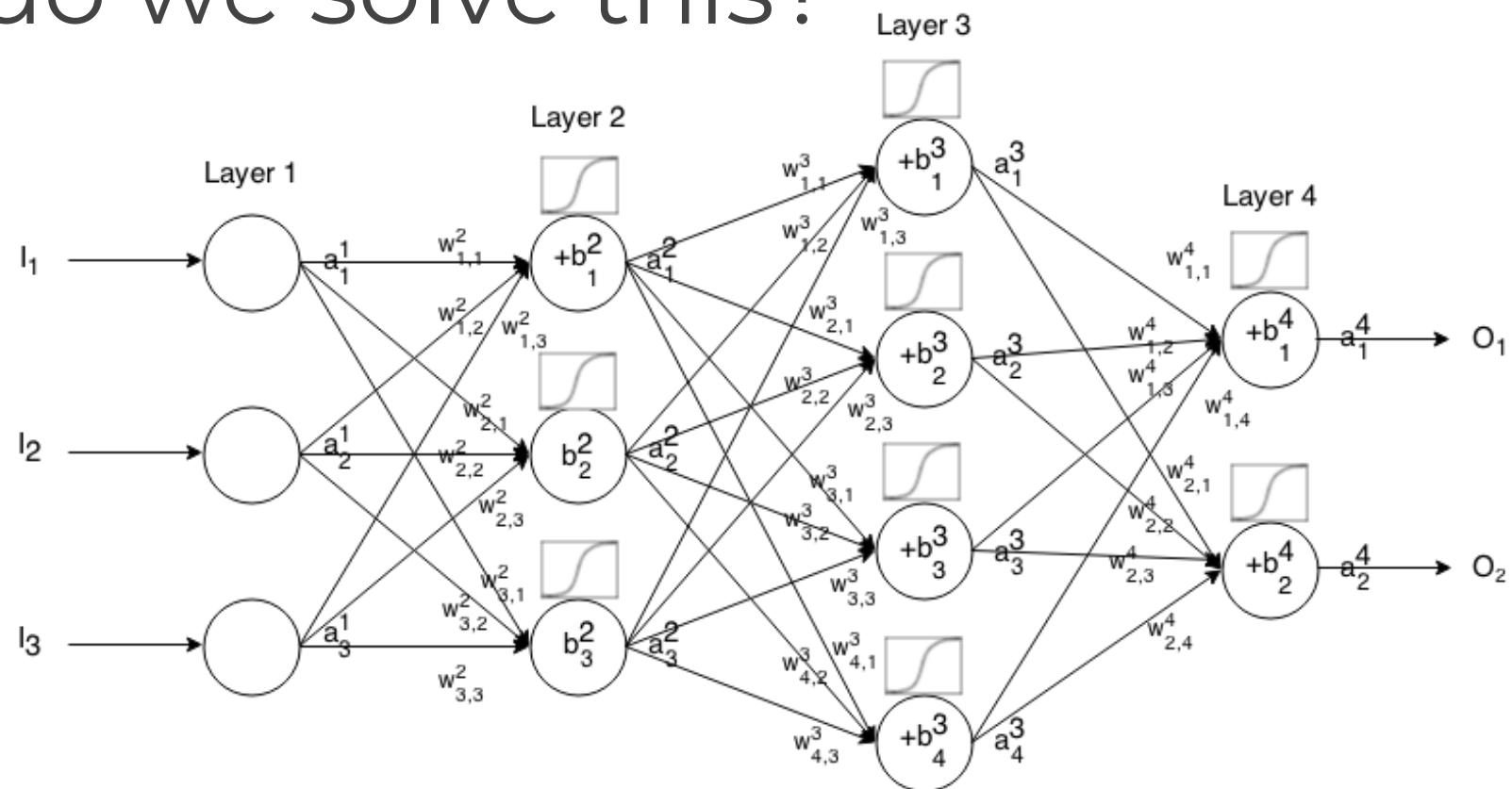
Deep Learning

- Here is a small network with all its parameters labeled:



Deep Learning

- That is a lot to calculate!
- How do we solve this?



Deep Learning

- In a real case, this means we have some cost function **C** dependent lots of weights!
 - **C(w₁,w₂,w₃,...,w_n)**
 - How do we figure out which weights lead us to the lowest cost?

Deep Learning

- For simplicity, let's imagine we only had one weight in our cost function w .
- We want to **minimize** our loss/cost (overall error).
- Which means we need to figure out what value of w results in the minimum of $C(w)$

Deep Learning

- The learning rate we showed in our illustrations was constant (each step size was equal)
- But we can be clever and adapt our step size as we go along.

Deep Learning

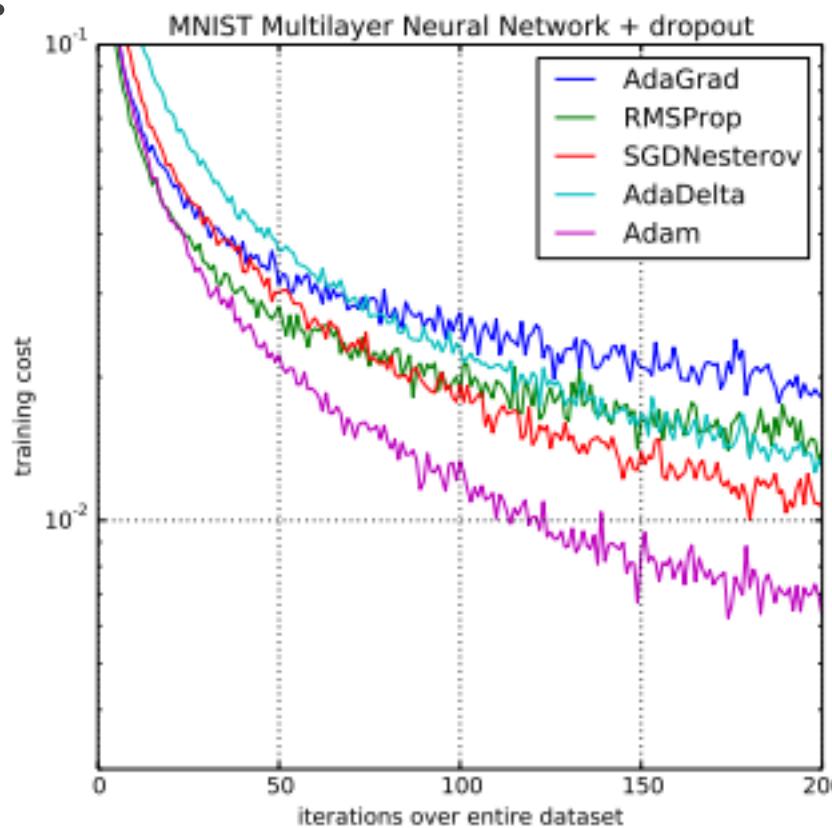
- We could start with larger steps, then go smaller as we realize the slope gets closer to zero.
- This is known as **adaptive gradient descent**.

Deep Learning

- In 2015, Kingma and Ba published their paper: “Adam: A Method for Stochastic Optimization”.
- Adam is a much more efficient way of searching for these minimums, so you will see us use it for our code!

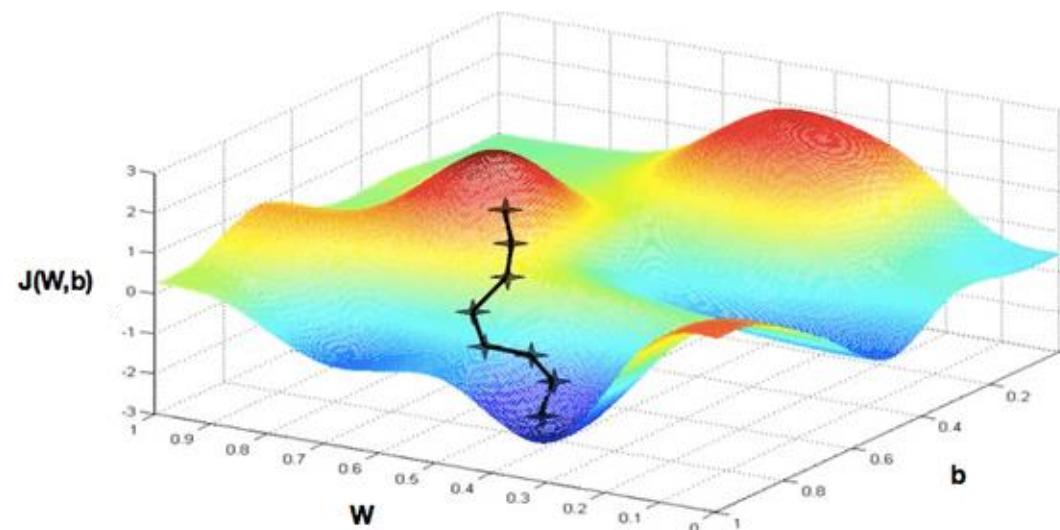
Deep Learning

- Adam versus other gradient descent algorithms:



Deep Learning

- Realistically we're calculating this descent in an n-dimensional space for all our weights.



Deep Learning

- When dealing with these N-dimensional vectors (tensors), the notation changes from **derivative** to **gradient**.
- This means we calculate $\nabla C(w_1, w_2, \dots, w_n)$

Deep Learning

- For classification problems, we often use the **cross entropy** loss function.
- The assumption is that your model predicts a probability distribution $p(y=i)$ for each class $i=1,2,\dots,C$.

Deep Learning

- For a binary classification this results in:
 $-(y \log(p) + (1 - y) \log(1 - p))$
- For **M** number of classes > 2

$$-\sum_{c=1}^M y_{o,c} \log(p_{o,c})$$

Deep Learning

- Review:
 - Cost Functions
 - Gradient Descent
 - Adam Optimizer
 - Quadratic Cost and Cross-Entropy

Deep Learning

- So far we understand how networks can take in input , effect that input with weights, biases, and activation functions to produce an estimated output.
- Then we learned how to evaluate that output.

Deep Learning

- The last thing we need to learn about theory is:
 - Once we get our cost/loss value, how do we actually go back and adjust our weights and biases?
- This is **backpropagation**, and it is what we are going to cover next!

Backpropagation

Deep Learning

- The last theory topic we will cover is **backpropagation**.
- We'll start by building an intuition behind backpropagation, and then we'll dive into the calculus and notation of backpropagation.

Deep Learning

- Fundamentally, we want to know how the cost function results changes with respect to the weights in the network, so we can update the weights to minimize the cost function

Deep Learning

- Let's begin with a very simple network, where each layer only has 1 neuron



Deep Learning

- Each input will receive a weight and bias



Deep Learning

- This means we have:
 - $C(w_1, b_1, w_2, b_2, w_3, b_3)$



Deep Learning

- We've already seen how this process propagates forward.
- Let's start at the end to see the backpropagation.



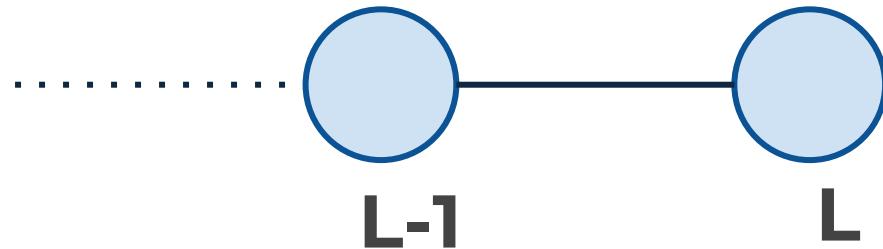
Deep Learning

- Let's say we have L layers, then our notation becomes:



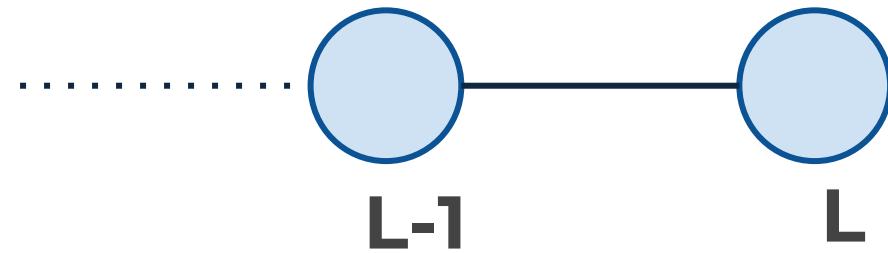
Deep Learning

- Focusing on these last two layers, let's define **$z = wx + b$**
- Then applying an activation function we'll state: **$a = \sigma(z)$**



Deep Learning

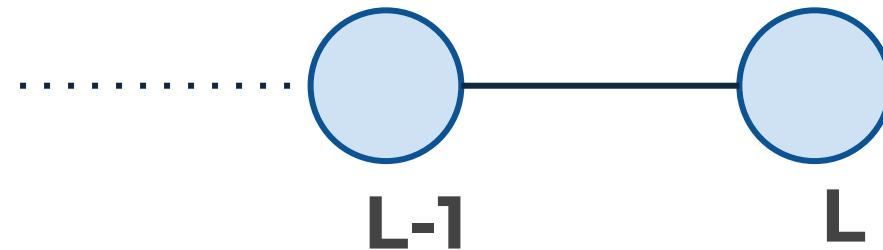
- This means we have:
 - $\mathbf{z}^L = \mathbf{w}^L \mathbf{a}^{L-1} + \mathbf{b}^L$



Deep Learning

- This means we have:

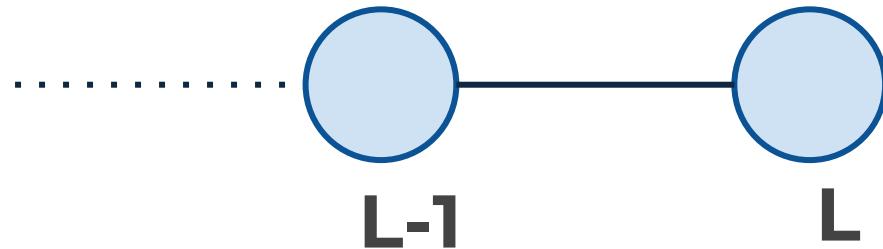
- $\mathbf{z}^L = \mathbf{w}^L \mathbf{a}^{L-1} + \mathbf{b}^L$
- $\mathbf{a}^L = \sigma(\mathbf{z}^L)$



Deep Learning

- This means we have:

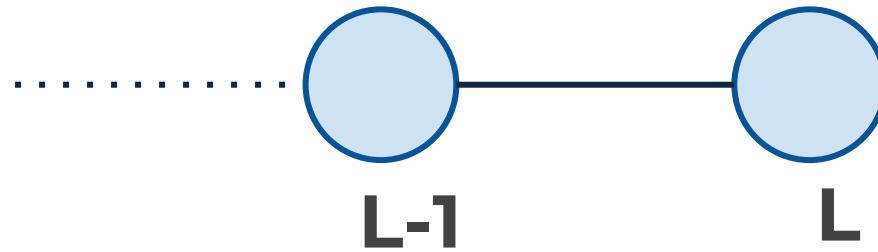
- $z^L = w^L a^{L-1} + b^L$
- $a^L = \sigma(z^L)$
- $C_0(\dots) = (a^L - y)^2$



Deep Learning

- We want to understand how sensitive is the cost function to changes in \mathbf{w} :

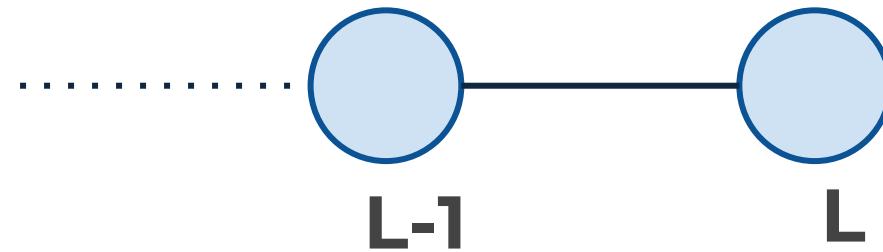
$$\frac{\partial C_0}{\partial w^L}$$



Deep Learning

- Using the relationships we already know along with the chain rule:

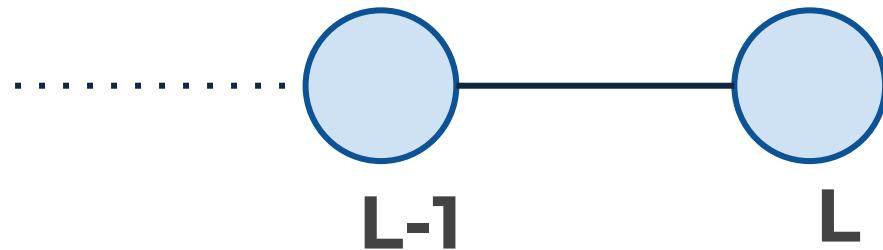
$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0}{\partial a^L}$$



Deep Learning

- We can calculate the same for the bias terms:

$$\frac{\partial C_0}{\partial b^L} = \frac{\partial z^L}{\partial b^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0}{\partial a^L}$$



Deep Learning

- The main idea here is that we can use the gradient to go back through the network and adjust our weights and biases to minimize the output of the error vector on the last output layer.

Deep Learning

- Using some calculus notation, we can expand this idea to networks with multiple neurons per layer.
- Hadamard Product

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

Deep Learning

- Given this notation and backpropagation, we have a few main steps to training neural networks.
- Note! You do not need to fully understand these intricate details to continue with the coding portions.

Deep Learning

- Step 1: Using input \mathbf{x} set the activation function \mathbf{a} for the input layer.
 - $\mathbf{z} = \mathbf{w}\mathbf{x} + \mathbf{b}$
 - $\mathbf{a} = \sigma(\mathbf{z})$
- This resulting \mathbf{a} then feeds into the next layer (and so on).

Deep Learning

- Step 2: For each layer, compute:
 - $\mathbf{z}^L = \mathbf{w}^L \mathbf{a}^{L-1} + \mathbf{b}^L$
 - $\mathbf{a}^L = \sigma(\mathbf{z}^L)$

Deep Learning

- Step 3: We compute our error vector:
 - $\delta^L = \nabla_a C \odot \sigma'(z^L)$

Deep Learning

- Step 3: We compute our error vector:
 - $\delta^L = \nabla_a C \odot \sigma'(z^L)$
 - $\nabla_a C = (a^L - y)$
 - **Expressing the rate of change of C with respect to the output activations**

Deep Learning

- Step 3: We compute our error vector:
 - $\delta^L = (a^L - y) \odot \sigma'(z^L)$

Deep Learning

- Step 3: We compute our error vector:
 - $\delta^L = (a^L - y) \odot \sigma'(z^L)$
- Now let's write out our error term for a layer in terms of the error of the next layer (since we're moving backwards).
- Font Note: lowercase **L**
- Font Note: Number **1**

Deep Learning

- Step 4: Backpropagate the error:
 - For each layer: $L-1, L-2, \dots$ we compute (note the lowercase L (l)):
 - $\delta^l = (\mathbf{w}^{l+1})^T \delta^{l+1} \odot \sigma'(\mathbf{z}^l)$
 - $(\mathbf{w}^{l+1})^T$ is the transpose of the weight matrix of $l+1$ layer

Deep Learning

- Step 4: Backpropagate the error:
 - This is the generalized error for any layer l :
 - $\delta^l = (\mathbf{w}^{l+1})^T \delta^{l+1} \odot \sigma'(\mathbf{z}^l)$
 - $(\mathbf{w}^{l+1})^T$ is the transpose of the weight matrix of $L+1$ layer

Deep Learning

- Step 4: When we apply the transpose weight matrix, $(\mathbf{w}^{l+1})^T$ we can think intuitively of this as moving the error backward through the network, giving us some sort of measure of the error at the output of the l th layer.

Deep Learning

- Step 4: We then take the Hadamard product $\odot \sigma'(z^l)$. This moves the error backward through the activation function in layer l , giving us the error δ^l in the weighted input to layer l .

Deep Learning

- The gradient of the cost function is given by:
 - For each layer: $L-1, L-2, \dots$ we compute

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

Deep Learning

- This then allows us to adjust the weights and biases to help minimize that cost function.
- Check out the external links for more details!

TensorFlow and Keras

Deep Learning

- Before we begin learning how to code our own neural networks, let's quickly clarify the differences between TensorFlow and Keras!

Deep Learning

- TensorFlow is an open-source deep learning library developed by Google, with TF 2.0 being officially released in late 2019.

Deep Learning

- TensorFlow has a large ecosystem of related components, including libraries like Tensorboard, Deployment and Production APIs, and support for various programming languages.

Deep Learning

- Keras is a high-level python library that can use a variety of deep learning libraries underneath, such as: TensorFlow, CNTK, or Theano.

Deep Learning

- TensorFlow 1.x had a complex python class system for building models, and due to the huge popularity of Keras, when TF 2.0 was released, TF adopted Keras as the official API for TF.

Deep Learning

- While Keras still also remains as a separate library from Tensorflow, it can also now officially be imported through TF, so there is now need to additionally install it.

Deep Learning

- The Keras API is easy to use and builds models by simply adding layers on top of each other through simple calls.
- Let's now explore the basics of the Keras API for TensorFlow!