

# *Machine Learning Basics*

*Grishma Sharma*

# Why “Learn”?

- *Machine learning is programming computers to optimize a performance criterion using example data or past experience.*
- There is no need to “learn” to calculate payroll
- Learning is used when:
  - Human expertise does not exist (navigating on Mars),
  - Humans are unable to explain their expertise (speech recognition)
  - Solution changes in time (routing on a computer network)
  - Solution needs to be adapted to particular cases (user biometrics)

# *What Is Machine Learning?*

- *Machine Learning is the science (and art) of programming computers so they can learn from data.*
- *[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.*

*Arthur Samuel, 1959*

And a more engineering-oriented one:

*A computer program is said to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its performance on  $T$ , as measured by  $P$ , improves with experience  $E$ .*

*Tom Mitchell, 1997*

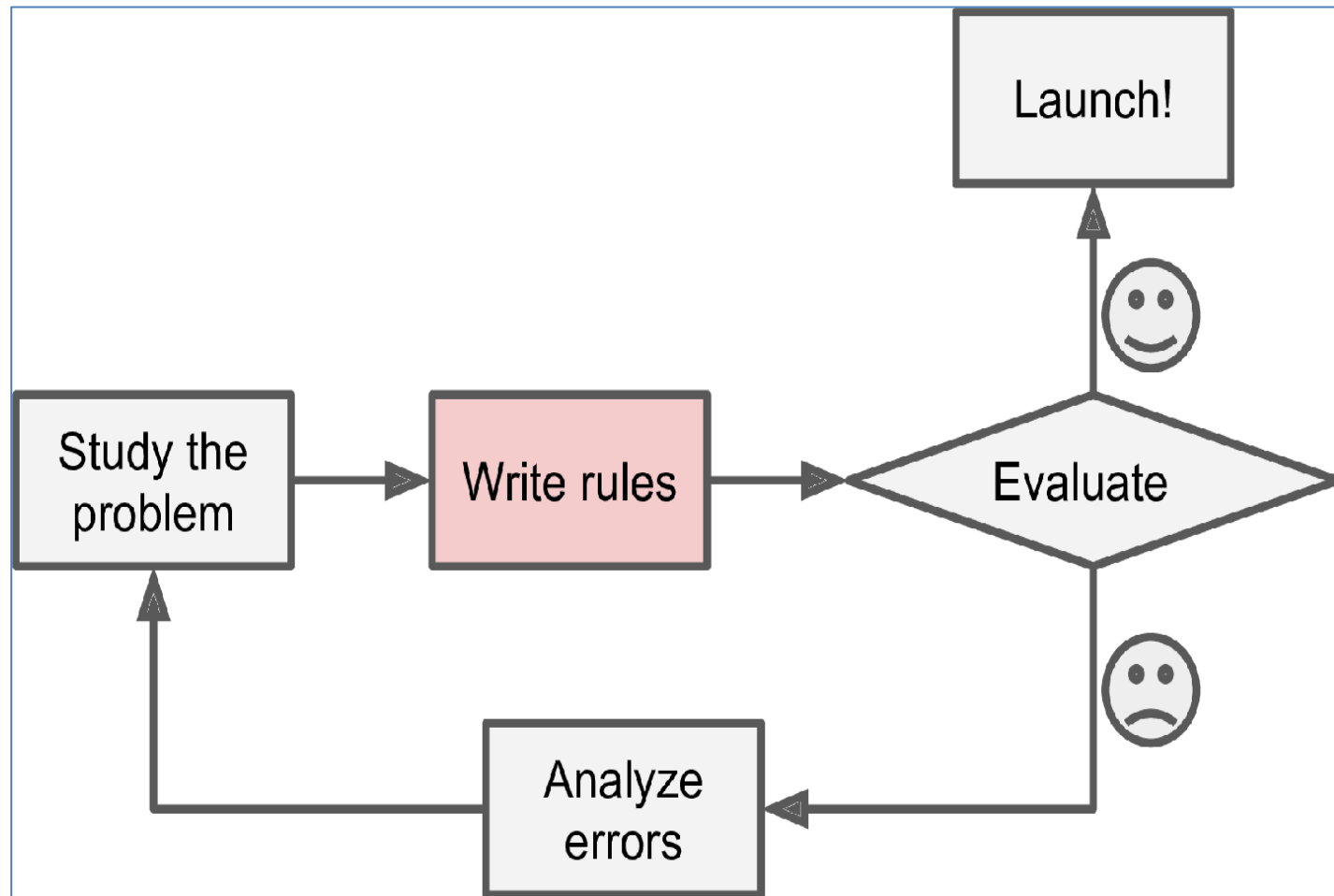
## *For example:*

- Your spam filter is a Machine Learning program ,
- given examples of spam emails (e.g., flagged by users) and examples of regular (nospam, also called “ham”) emails, can learn to flag spam.
- The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample).
- In this case, the task  $T$  is to flag spam for new emails, the experience  $E$  is the training data, and the performance measure  $P$  needs to be defined;
- for example, you can use the ratio of correctly classified emails. This particular performance measure is called accuracy, and it is often used in classification tasks.

# Why Use Machine Learning?

*Consider how you would write a spam filter using traditional programming techniques*

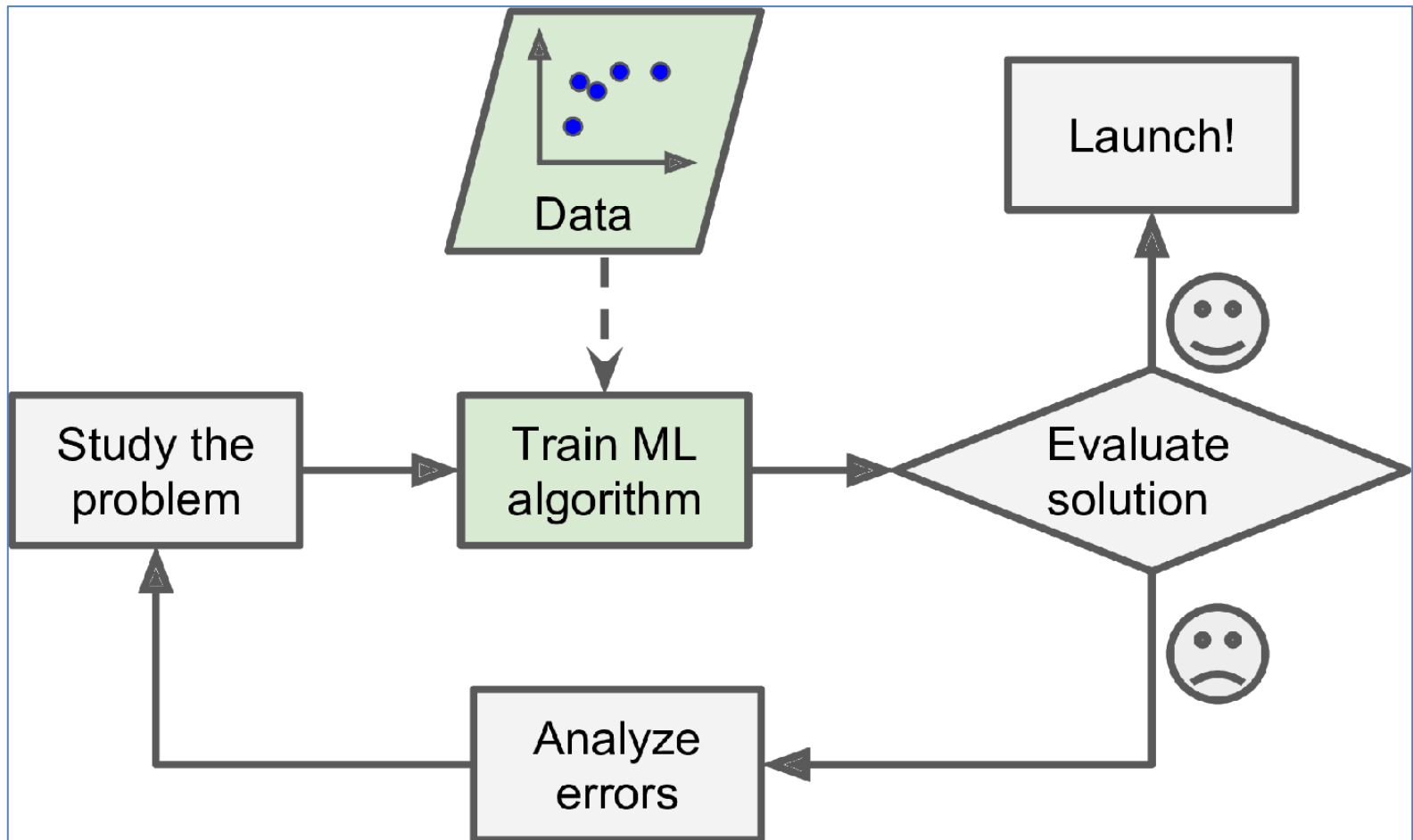
- First you would consider what spam typically looks like. You might notice that some words or phrases (such as “4U,” “credit card,” “free,” and “amazing”) tend to come up a lot in the subject line. Perhaps you would also notice a few other patterns in the sender’s name, the email’s body, and other parts of the email.
- You would write a detection algorithm for each of the patterns that you noticed, and your program would flag emails as spam if a number of these patterns were detected.
- You would test your program and repeat steps 1 and 2 until it was good enough to launch.



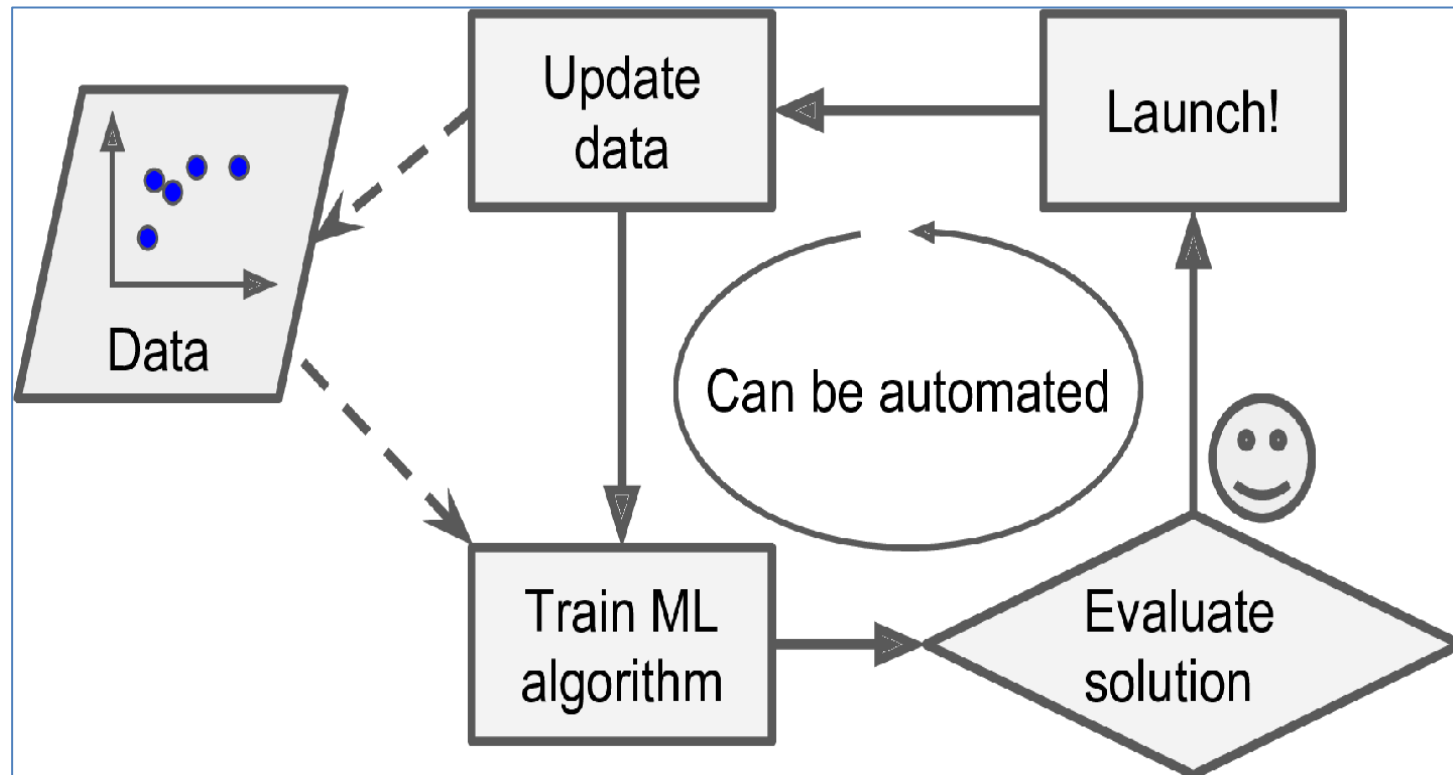
*The traditional approach*

Since the problem is difficult, your program will likely become a long list of complex rules—pretty hard to maintain.

- In contrast, a spam filter based on Machine Learning techniques automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples. The program is much shorter, easier to maintain, and most likely more accurate.
- What if spammers notice that all their emails containing “4U” are blocked? They might start writing “For U” instead.
- A spam filter using traditional programming techniques would need to be updated to flag “For U” emails. If spammers keep working around your spam filter, you will need to keep writing new rules forever.
- In contrast, a spam filter based on Machine Learning techniques automatically notices that “For U” has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention



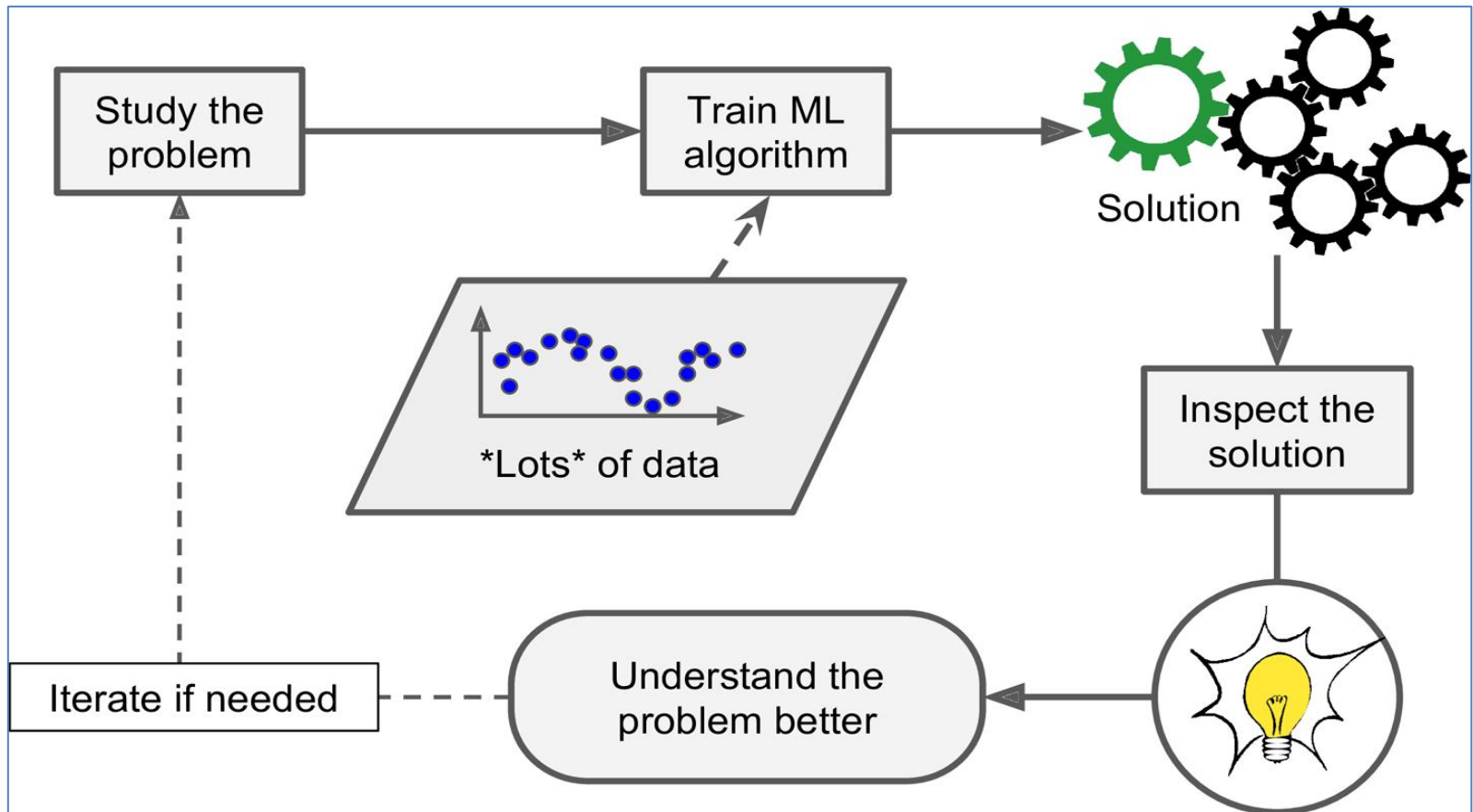
*The Machine Learning approach*



*Automatically adapting to change*

- Another area where Machine Learning shines is for problems that either are too complex for traditional approaches or have no known algorithm.
- For example, consider speech recognition. Say you want to start simple and write a program capable of distinguishing the words “one” and “two.”
- You might notice that the word “two” starts with a high-pitch sound (“T”), so you could hardcode an algorithm that measures high-pitch sound intensity and use that to distinguish ones and twos.
- but obviously this technique will not scale to thousands of words spoken by millions of very different people in noisy environments and in dozens of languages.
- The best solution (at least today) is to write an algorithm that learns by itself, given many example recordings for each word.

- Finally, Machine Learning can help humans learn. ML algorithms can be inspected to see what they have learned (although for some algorithms this can be tricky).
- For instance, once a spam filter has been trained on enough spam, it can easily be inspected to reveal the list of words and combinations of words that it believes are the best predictors of spam.
- Sometimes this will reveal unsuspected correlations or new trends, and thereby lead to a better understanding of the problem.
- Applying ML techniques to dig into large amounts of data can help discover patterns that were not immediately apparent. **This is called data mining.**



*Machine Learning can help humans learn*

***To summarize, Machine Learning is great for:***

- Problems for which existing solutions require a lot of fine-tuning or long lists of rules: one Machine Learning algorithm can often simplify code and perform better than the traditional approach.
- Complex problems for which using a traditional approach yields no good solution: the best Machine Learning techniques can perhaps find a solution.
- Fluctuating environments: a Machine Learning system can adapt to new data.
- Getting insights about complex problems and large amounts of data.

## ***Examples of Applications:***

Let's look at some concrete examples of Machine Learning tasks, along with the techniques that can tackle them:

- Analyzing images of products on a production line to automatically classify them. This is **image classification**, typically performed using convolutional neural networks (CNNs).
- **Detecting tumors in brain scans**  
This is semantic segmentation, where each pixel in the image is classified (as we want to determine the exact location and shape of tumors), typically using CNNs as well.
- **Automatically classifying news articles**  
This is natural language processing (NLP), and more specifically text classification, which can be tackled using recurrent neural networks (RNNs), CNNs, or Transformers.
- Automatically flagging offensive comments on discussion forums. This is also **text classification**, using the same NLP tools.

- **Summarizing long documents** automatically. This is a branch of NLP called text summarization, again using the same tools.
- Creating a **chatbot** or a **personal assistant**. This involves many NLP components, including natural language understanding (NLU) and question-answering modules.
- **Forecasting your company's revenue next year**, based on many performance metrics. This is a regression task (i.e., predicting values) that may be tackled using any regression model, such as a Linear Regression or Polynomial Regression model, a regression SVM, a regression Random Forest, or an artificial neural network. If you want to take into account sequences of past performance metrics, you may want to use RNNs, CNNs, or Transformers.
- **Making your app react to voice commands**  
This is speech recognition, which requires processing audio samples: since they are long and complex sequences, they are typically processed using RNNs, CNNs, or Transformers.
- **Detecting credit card fraud**  
This is anomaly detection.
- **Segmenting clients based on their purchases** so that you can design a different marketing strategy for each segment. **This is clustering.**

- Representing a complex, high-dimensional dataset in a clear and insightful diagram. This is **data visualization**; often involving dimensionality reduction techniques.
- **Recommending a product** that a client may be interested in, based on past purchases  
This is a recommender system. One approach is to feed past purchases (and other information about the client) to an artificial neural network, and get it to output the most likely next purchase. This neural net would typically be trained on past sequences of purchases across all clients.
- **Building an intelligent bot for a game**  
This is often tackled using Reinforcement Learning, which is a branch of Machine Learning that trains agents (such as bots) to pick the actions that will maximize their rewards over time (e.g., a bot may get a reward every time the player loses some life points), within a given environment (such as the game). The famous AlphaGo program that beat the world champion at the game of Go was built using RL.

## Common Terminologies in Machine Learning

Vector  
Feature

It is an  $n$ -dimensional vector of numerical features that represent some object.

Samples

They are the items to process.

Feature  
Space

It refers to the collections of features that are used to characterize your data.

Labeled  
Data

It is the data with known classification results.

## *Types of Machine Learning Systems:*

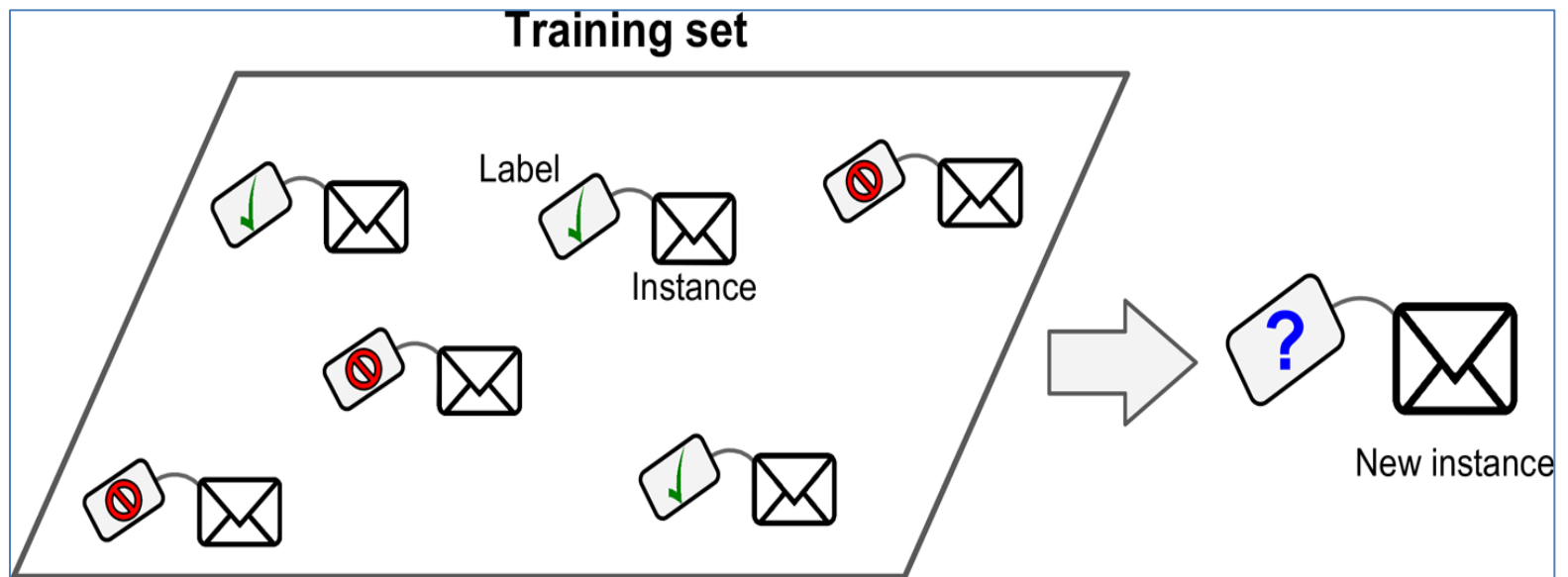
- There are so many different types of Machine Learning systems that it is useful to classify them in broad categories, based on the following criteria:
- Whether or not they are trained with human supervision (**supervised, unsupervised, semi supervised, and Reinforcement Learning**)
- Whether or not they can learn incrementally on the fly (online versus batch learning)
- Whether they work by simply comparing new data points to known data points, or instead by detecting patterns in the training data and building a predictive model, much like scientists do (instance-based versus model-based learning)
- These criteria are not exclusive; you can combine them in any way you like. For example, a state-of-the-art spam filter may learn on the fly using a deep neural network model trained using examples of spam and ham; this makes it an online, model-based, supervised learning system.

## ***Supervised/Unsupervised Learning:***

- Machine Learning systems can be classified according to the amount and type of supervision they get during training.
- There are four major categories: **supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning.**

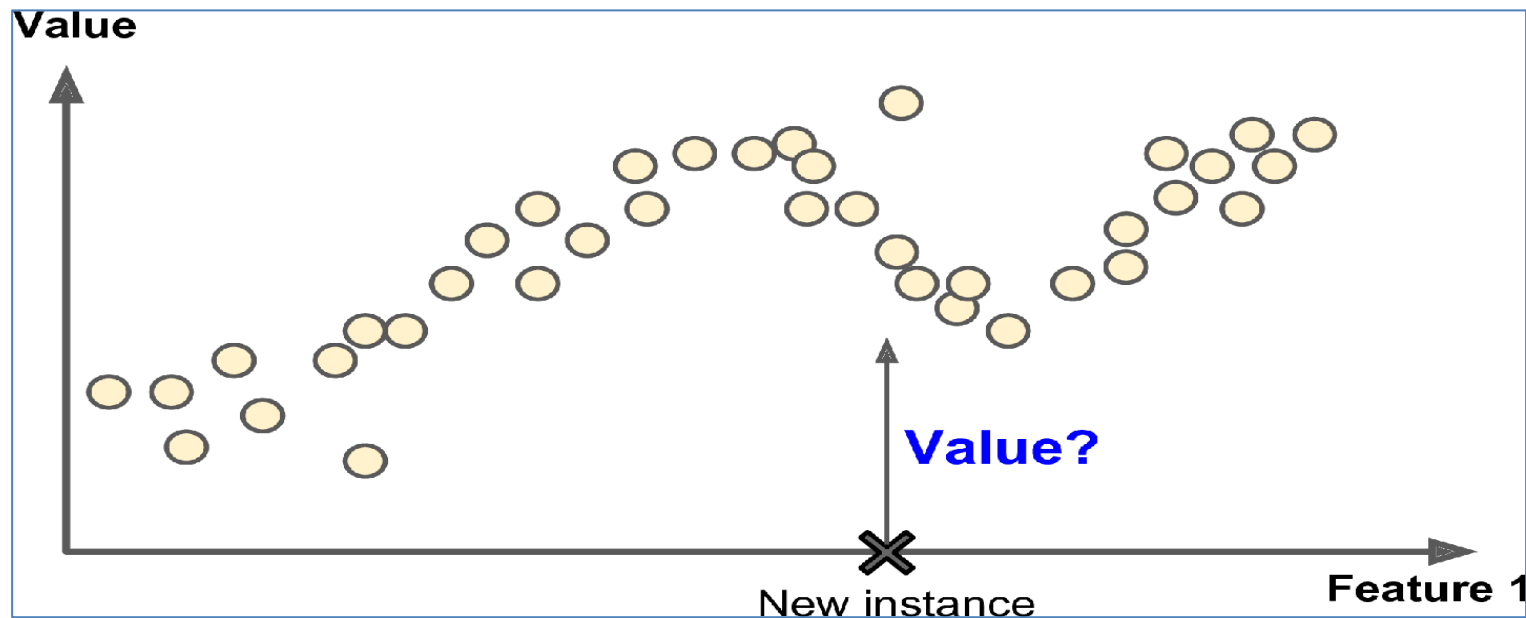
### **1. Supervised Learning**

- In supervised learning, the training set you feed to the algorithm includes the desired solutions, called labels.



*A labeled training set for spam classification (an example of supervised learning)*

- A typical supervised learning task is **classification**. The spam filter is a good example of this: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails.
- Another typical task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc.) called predictors. This sort of task is called **regression**.
- To train the system, you need to give it many examples of cars, including both their predictors and their labels (i.e., their prices).



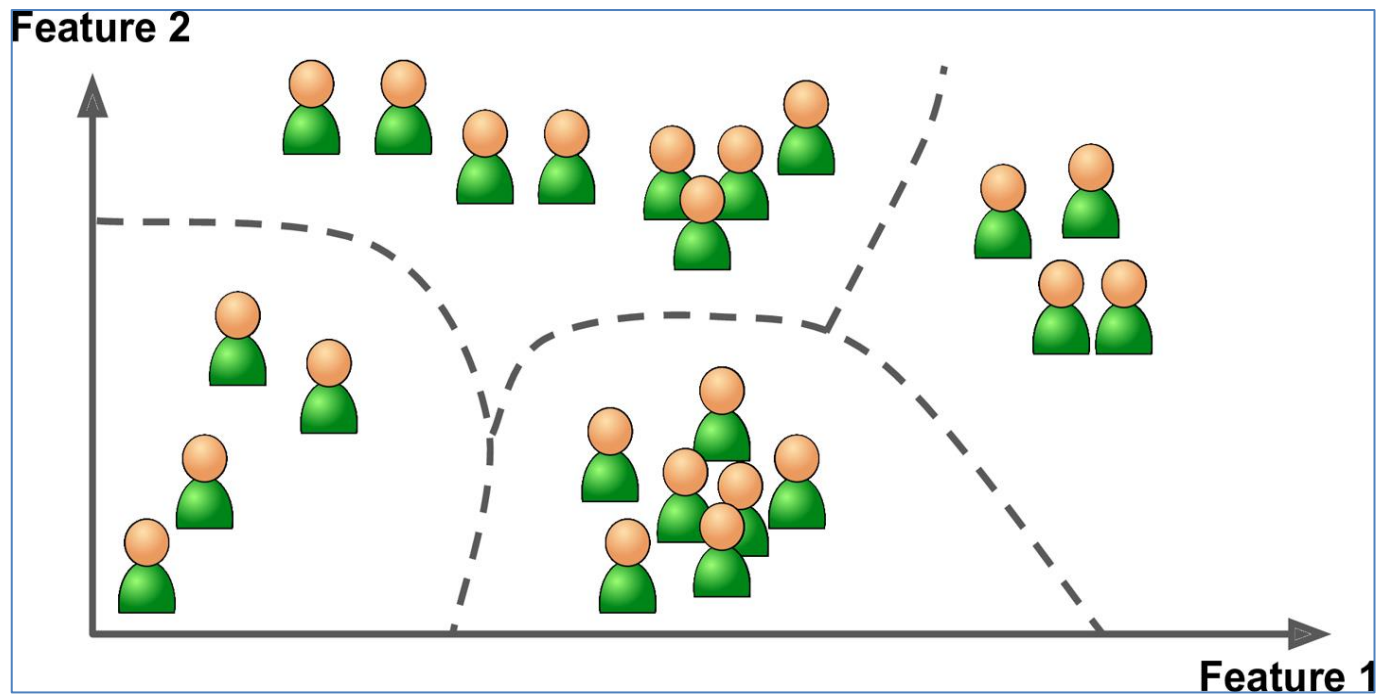
*Figure : A regression problem: predict a value, given an input feature (there are usually multiple input features, and sometimes multiple output values)*

## *Supervised learning algorithms:*

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks

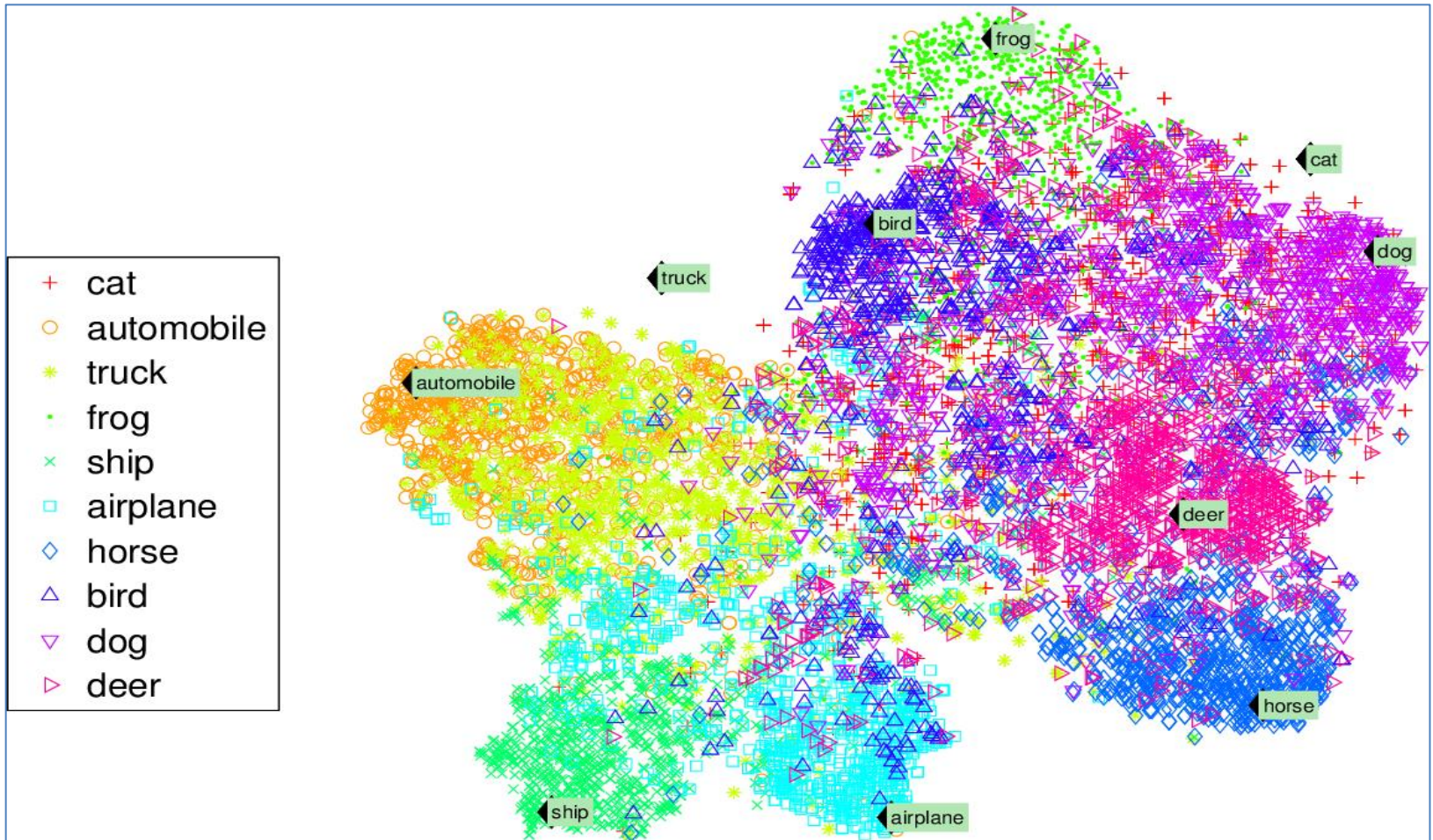
## 2. *Unsupervised Learning*

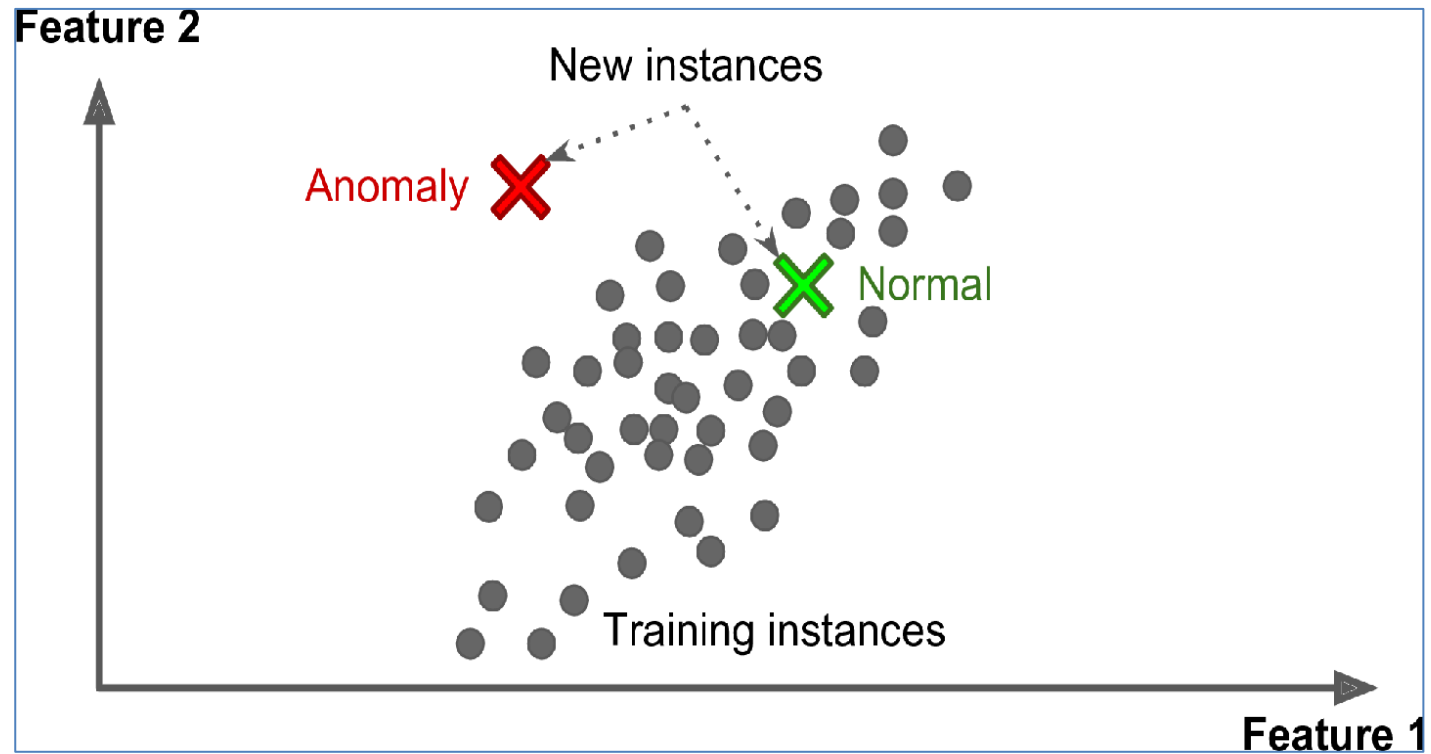
- In unsupervised learning, as you might guess, the training data is unlabeled . The system tries to learn without a teacher.
- For example, say you have a lot of data about your blog's visitors. You may want to run a clustering algorithm to try to detect groups of similar visitors. At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help.
- For example, it might notice that 40% of your visitors are males who love comic books and generally read your blog in the evening, while 20% are young sci-fi lovers who visit during the weekends. If you use a hierarchical clustering algorithm, it may also subdivide each group into smaller groups. This may help you target your posts for each group.



***Figure : Clustering***

- Visualization algorithms are also good examples of unsupervised learning algorithms: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted.





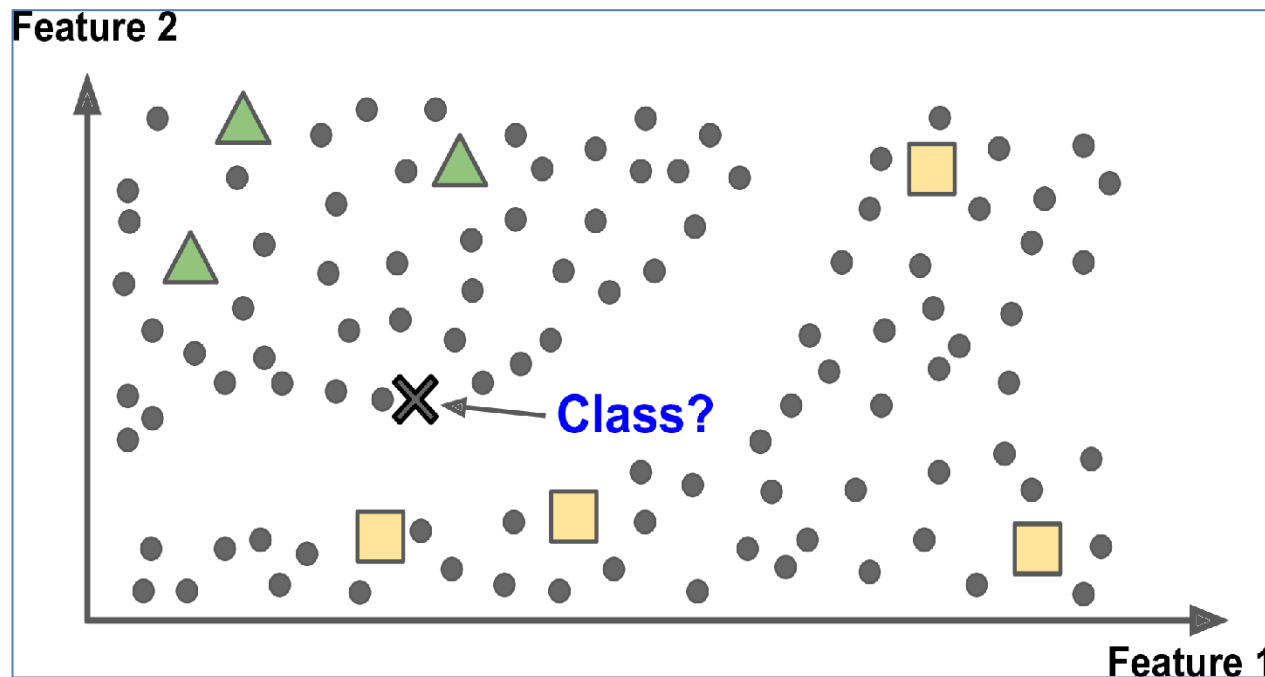
**Figure : Anomaly detection**

# *Unsupervised learning algorithms*

- Clustering
- K-Means
- DBSCAN
- Hierarchical Cluster Analysis (HCA)
- Anomaly detection and novelty detection
- One-class SVM
- Isolation Forest
- Visualization and dimensionality reduction
- Principal Component Analysis (PCA)
- Kernel PCA
- Locally Linear Embedding (LLE)
- t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning
- Apriori
- Eclat

### 3. Semisupervised Learning

Since labeling data is usually time-consuming and costly, you will often have plenty of unlabeled instances, and few labeled instances. Some algorithms can deal with data that's partially labeled. This is called *semisupervised learning*.

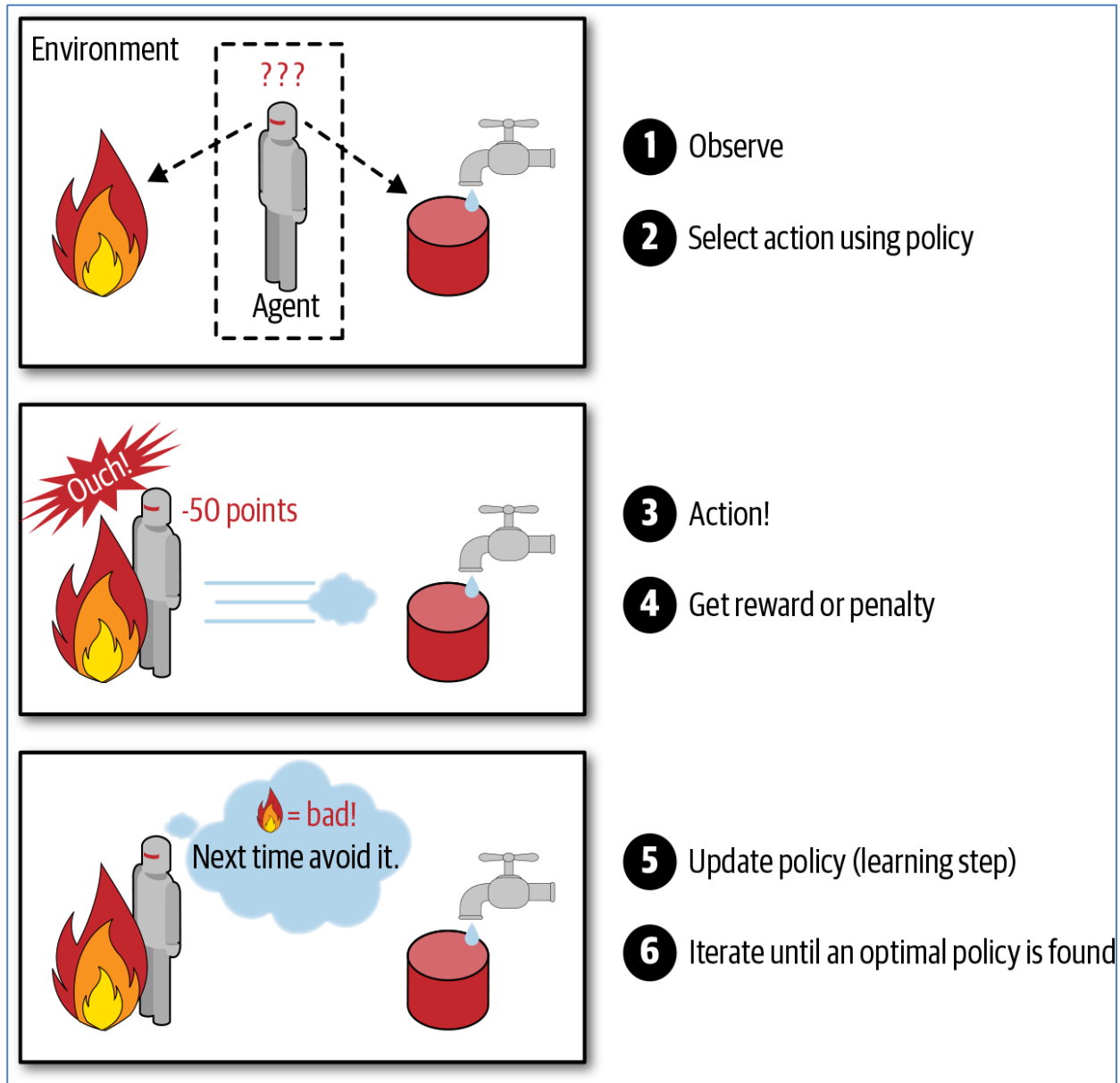


*Figure: Semisupervised learning with two classes (triangles and squares): the unlabeled examples (circles) help classify a new instance (the cross) into the triangle class rather than the square class, even though it is closer to the labeled squares*

- Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person A shows up in photos 1, 5, and 11, while another person B shows up in photos 2, 5, and 7.
- This is the unsupervised part of the algorithm (clustering). Now all the system needs is for you to tell it who these people are. Just add one label per person and it is able to name everyone in every photo, which is useful for searching photos.

## 4. Reinforcement Learning

- Reinforcement Learning is a very different beast. The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards).
- It must then learn by itself, what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.
- For example, many robots implement Reinforcement Learning algorithms to learn how to walk. DeepMind's AlphaGo program is also a good example of Reinforcement Learning: it made the headlines in May 2017 when it beat the world champion Ke Jie at the game of Go.
- It learned its winning policy by analyzing millions of games, and then playing many games against itself. Note that learning was turned off during the games against the champion; AlphaGo was just applying the policy it had learned.



**Figure: Reinforcement Learning**

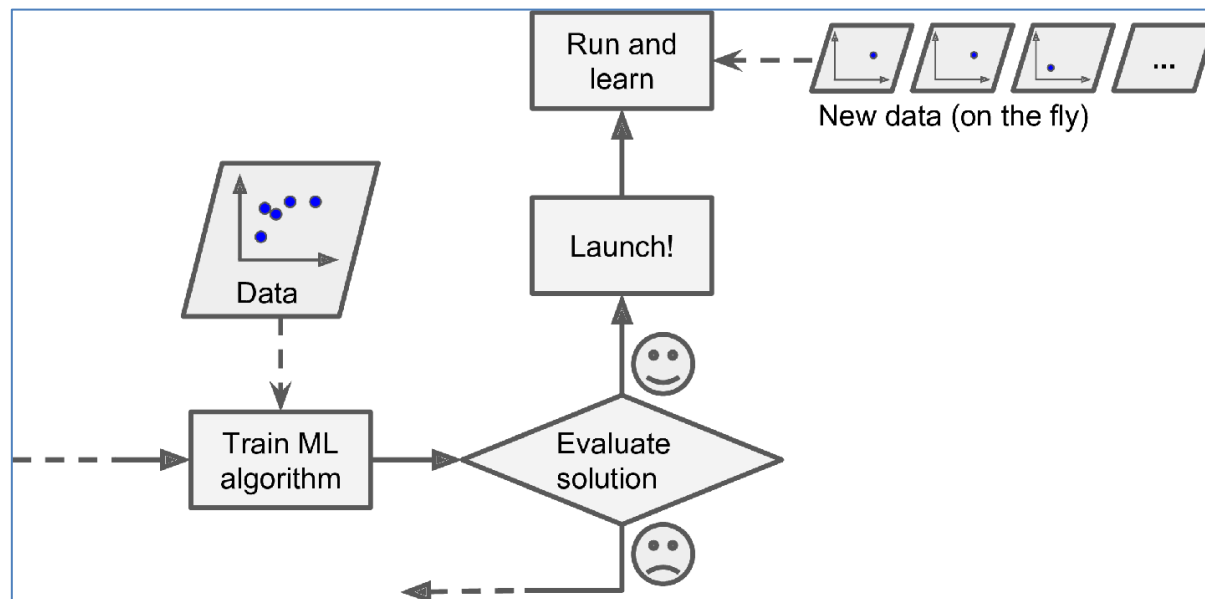
## ***5. Batch and Online Learning***

- Another criterion used to classify Machine Learning systems is whether or not the system can learn incrementally from a stream of incoming data.

### ***Batch Learning***

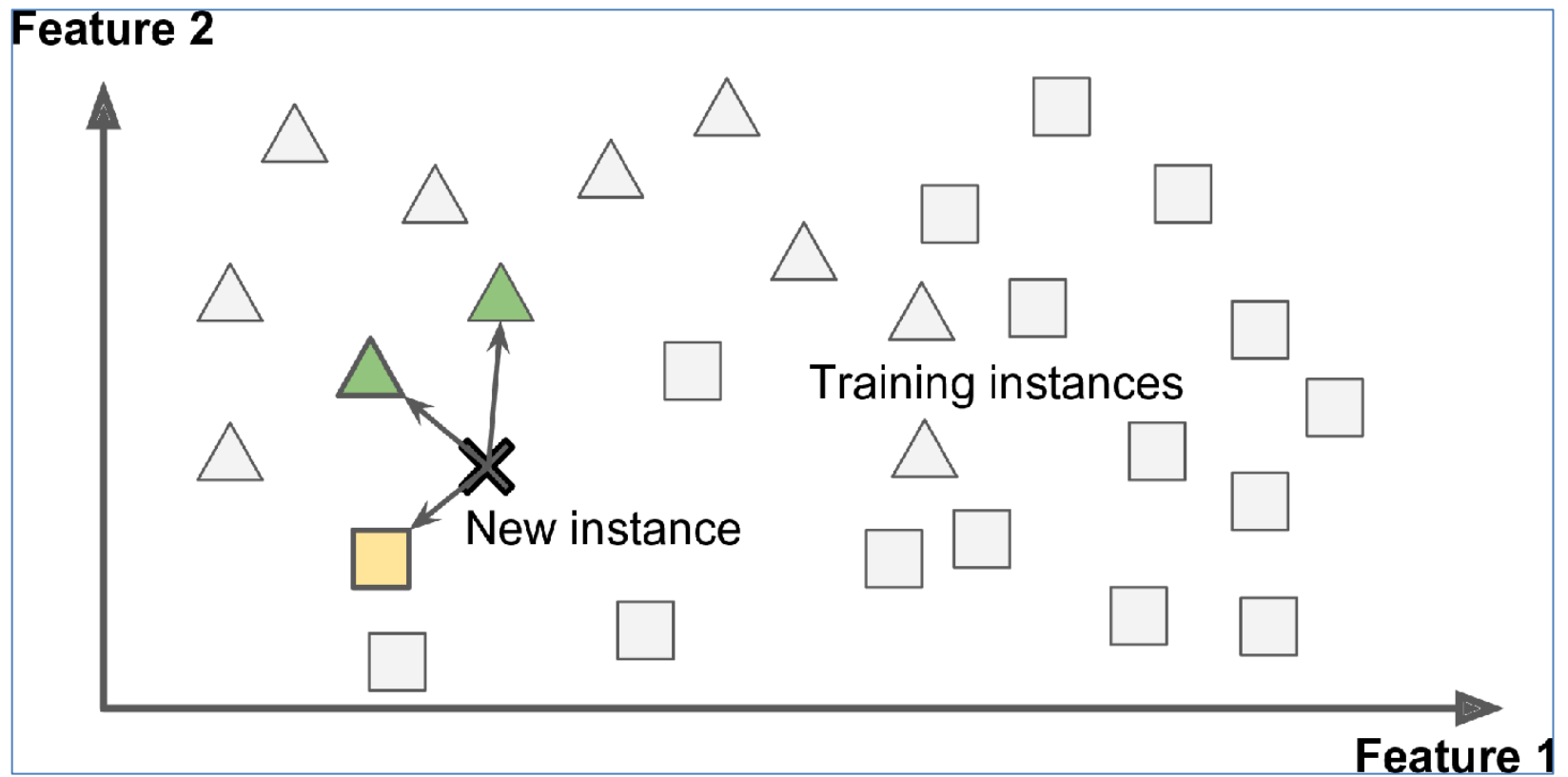
- In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline.
- First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning.

- In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or in small groups called mini-batches. Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives.
- Online learning is great for systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously.
- It is also a good option if you have limited computing resources: once an online learning system has learned about new data instances, it does not need them anymore, so you can discard them (unless you want to be able to roll back to a previous state and “replay” the data). This can save a huge amount of space.



## ***6. Instance-Based Versus Model-Based Learning***

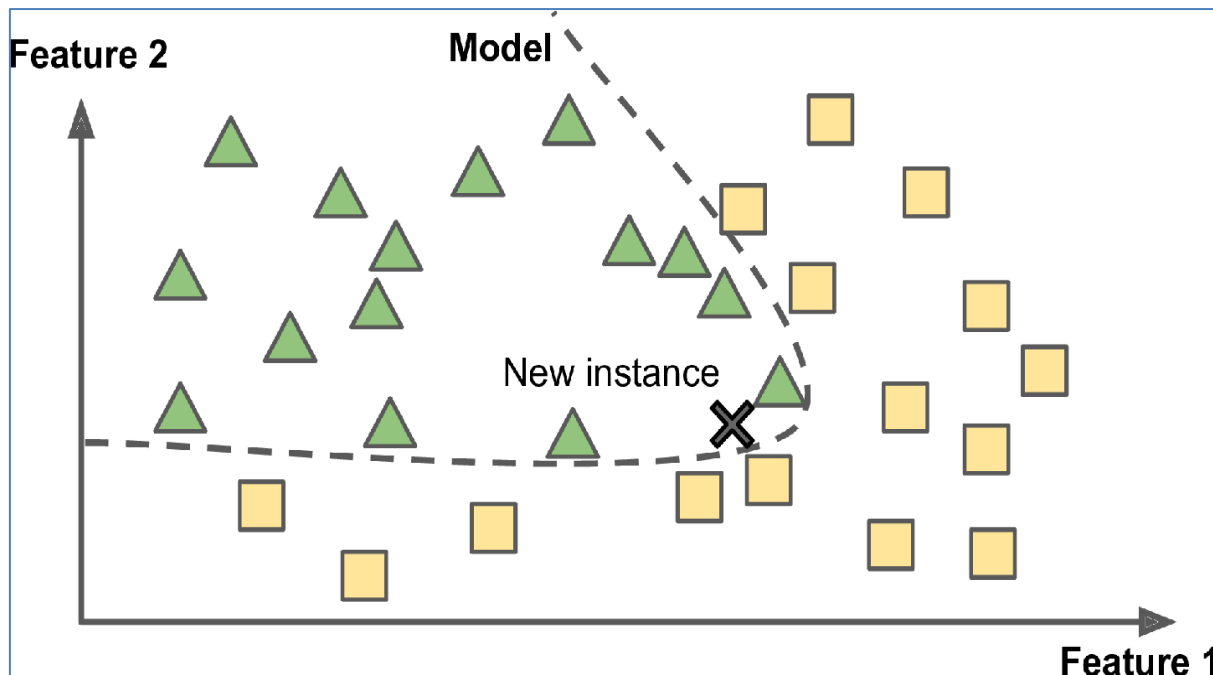
- One more way to categorize Machine Learning systems is by how they generalize. Most Machine Learning tasks are about making predictions.
- This means that given a number of training examples, the system needs to be able to make good predictions for (generalize to) examples it has never seen before. Having a good performance measure on the training data is good, but insufficient; the true goal is to perform well on new instances.
- There are two main approaches to generalization: instance-based learning and model-based learning.



***Figure: Instance-based learning***

## ***Model-Based Learning***

- Another way to generalize from a set of examples is to build a model of these examples and then use that model to make predictions. This is called model-based learning.



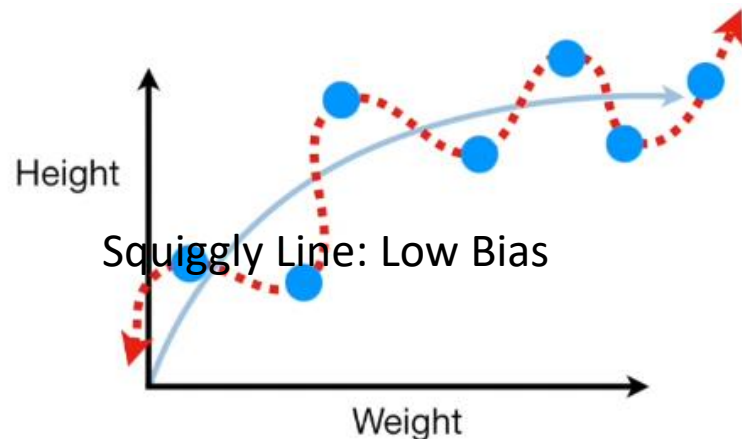
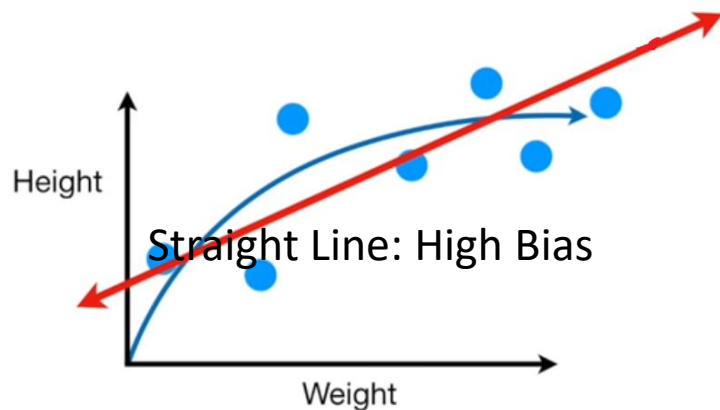
***Figure : Model-based learning***

## *Main Challenges of Machine Learning*

- Insufficient Quantity of Training Data
- Poor-Quality Data
- Irrelevant Features
- Overfitting the Training Data:
- Underfitting the Training Data:
- Testing and Validating:
- Hyperparameter Tuning and Model Selection:
- Data Mismatch

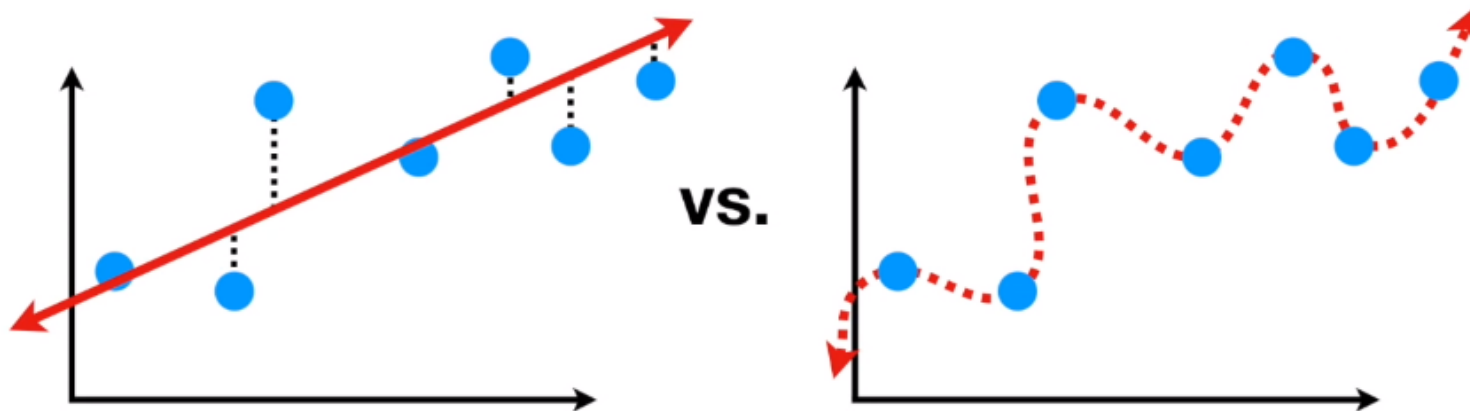
# ***Bias and Variance***

- The inability of a ML algorithm to capture a true relationship is called a bias



# ***Bias and Variance***

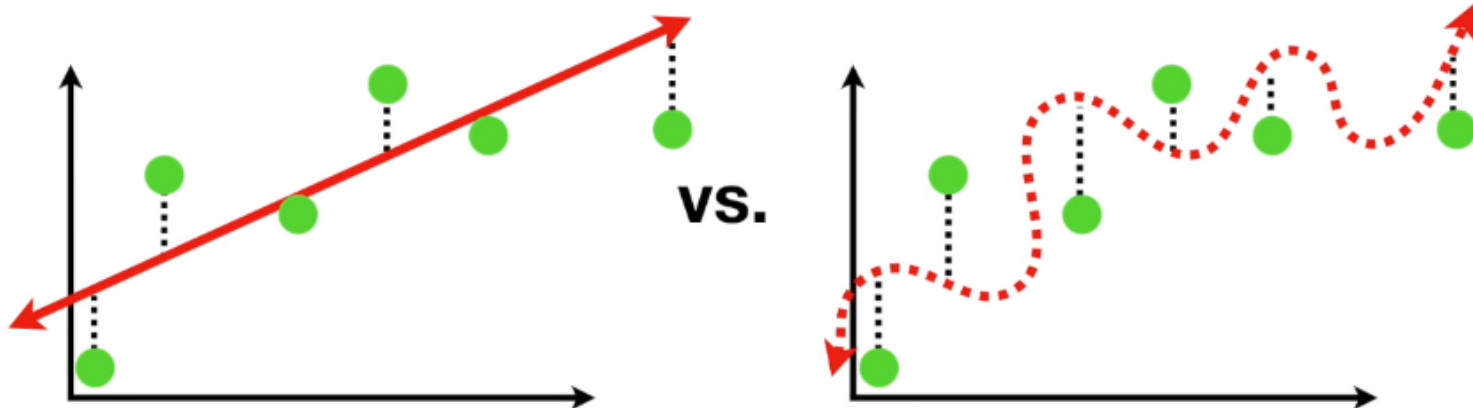
In the contest to see whether the **Straight Line** fits the **training set** better than the **Squiggly Line**...



- Find out the sum of squared distances of the training samples from the fitting line
- **The squiggly line wins** (sum of squared distances = 0)

# Bias and Variance

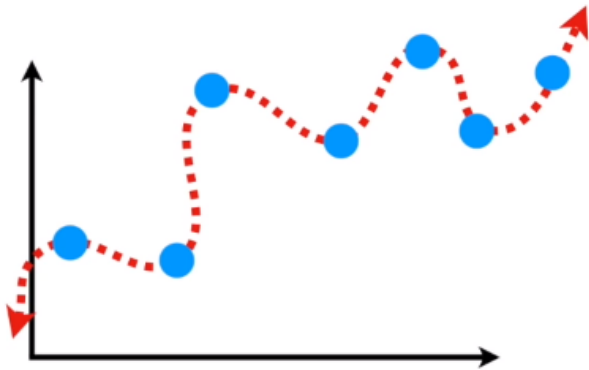
In the contest to see whether the **Straight Line** fits the **testing set** better than the **Squiggly Line**...



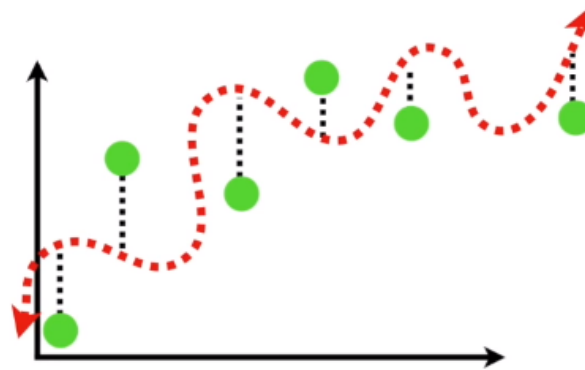
- Find out the sum of squared distances of the testing samples from the fitting line
- **The straight line wins** (sum of squared distances of St. line < squiggly line)

# *Bias and Variance*

In Machine Learning lingo, the difference in fits between data sets is called **Variance**.

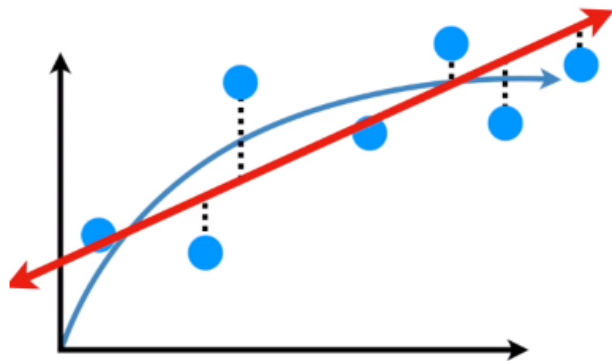


Low Bias

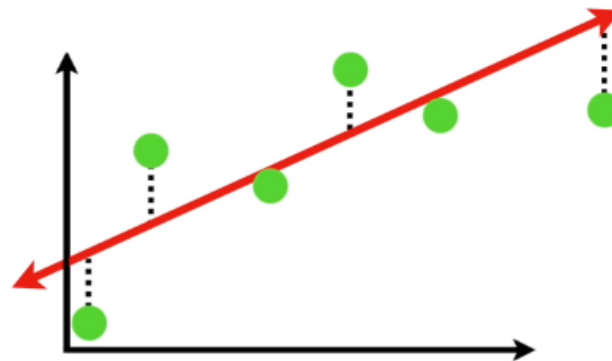


High Variance

Over Fitting



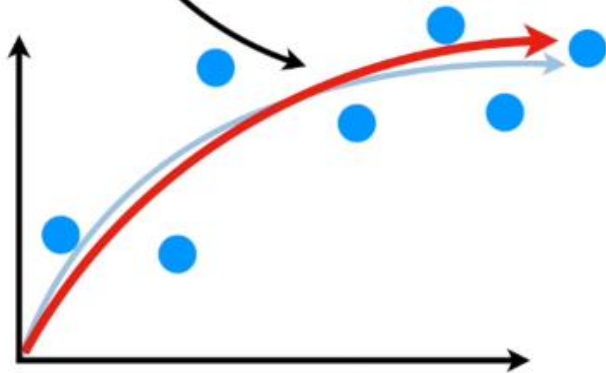
High Bias



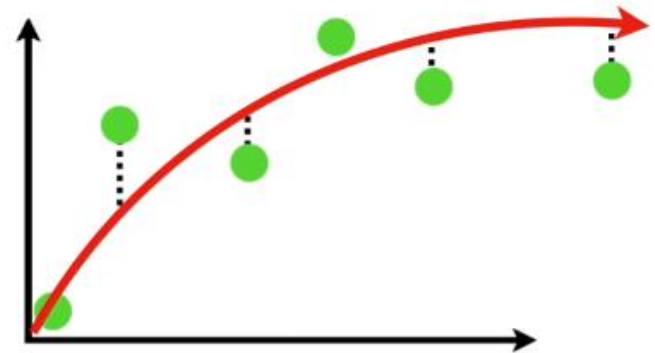
Low Variance

# Bias and Variance

In machine learning, the ideal algorithm has **low bias** and can accurately model the true relationship...



...and it has **low variability**, by producing consistent predictions across different datasets.



Find a sweet spot between the simple model and a complex model to have low bias and low variability

# *Over Fitting & Under Fitting*

- **Overfitting**

- ✓ occurs when a statistical model or machine learning algorithm captures the noise of the data
- ✓ Intuitively, overfitting occurs when the model or the algorithm fits the data too well
- ✓ Specifically, overfitting occurs if the model or algorithm shows low bias but high variance
- ✓ Overfitting is often a result of an excessively complicated model
- ✓ It happens when we try fitting lots and lots of training data
- ✓ It can be prevented by fitting multiple models and using validation or cross-validation to compare their predictive accuracies on test data.

# *Over Fitting & Under Fitting*

- **Underfitting**

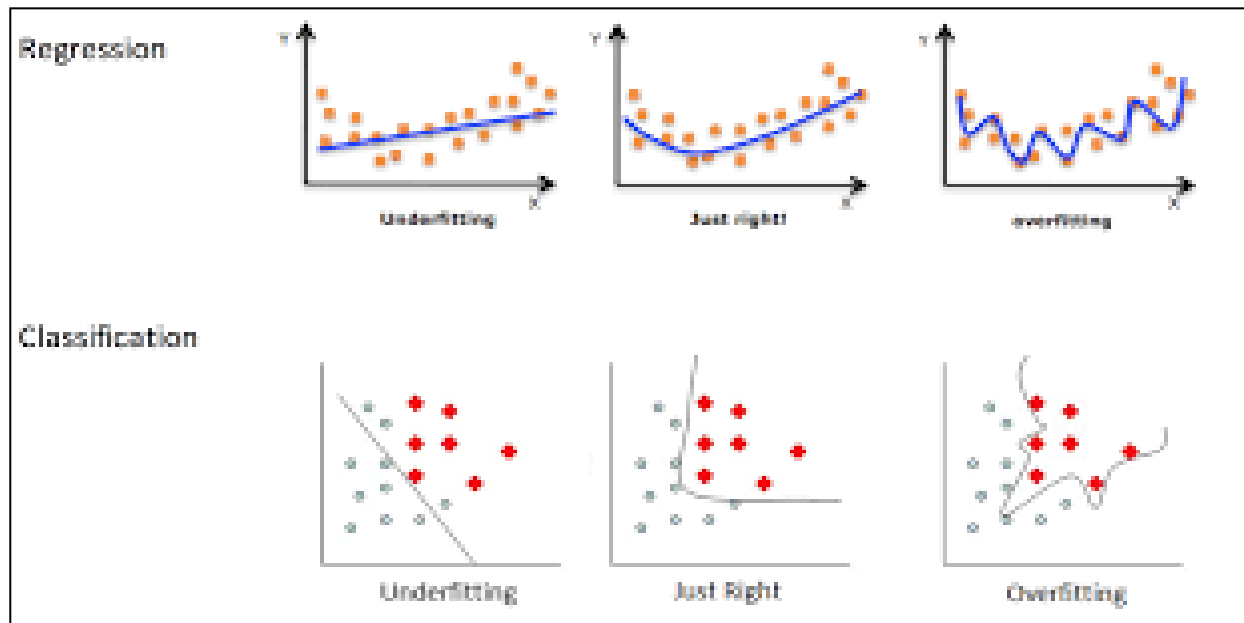
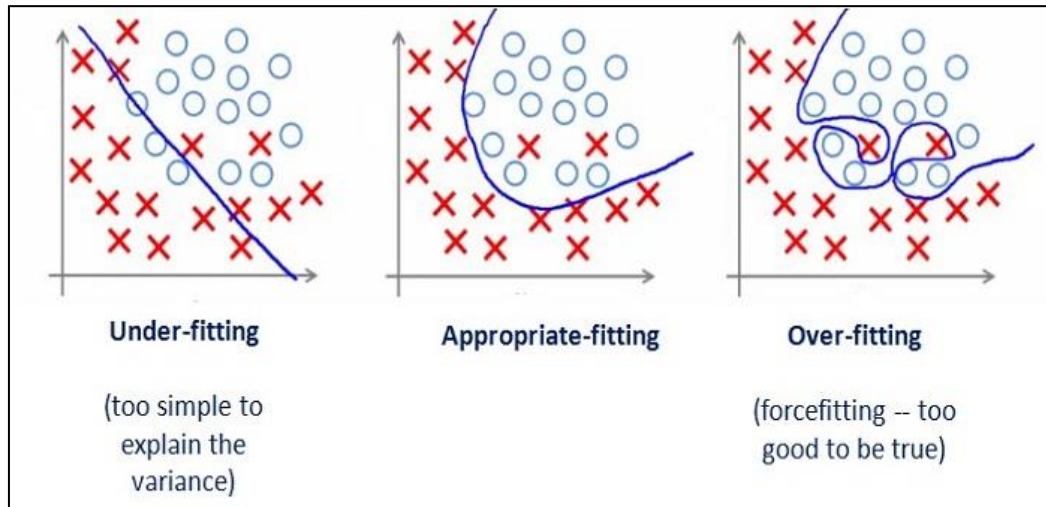
- ✓ occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data
- ✓ Intuitively, underfitting occurs when the model or the algorithm does not fit the data well enough
- ✓ Specifically, underfitting occurs if the model or algorithm shows low variance but high bias
- ✓ Underfitting is often a result of an excessively simple model
- ✓ It happens when we have very less data for training

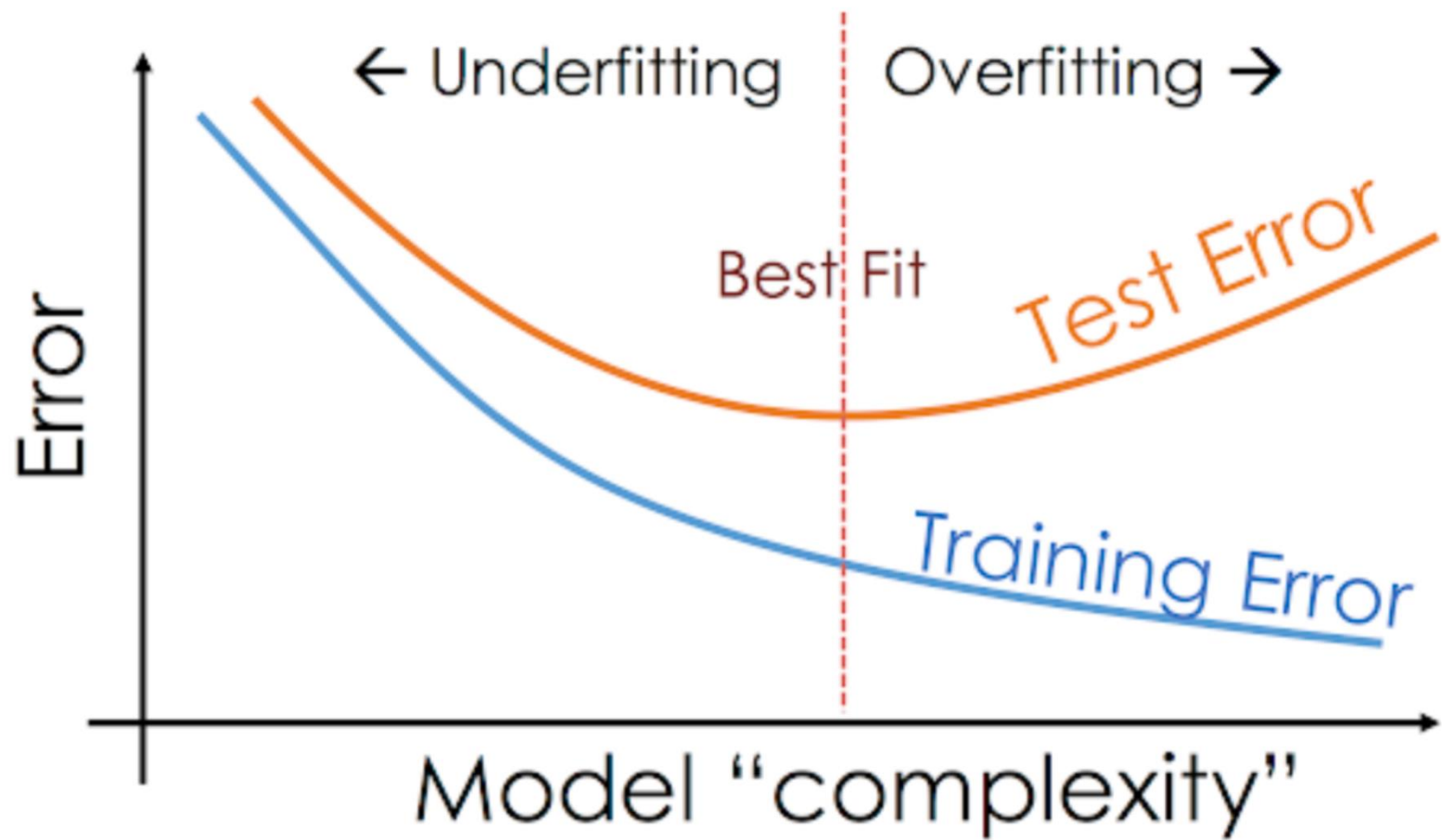
- Both overfitting and underfitting lead to poor predictions on new data sets.

- A model that overfits or underfits is not able to **generalize** well

**Generalization** refers to your model's ability to adapt properly to new, previously unseen data, drawn from the same distribution as the one used to create the model

# Over Fitting & Under Fitting





# ***Training, Validation and Testing***

## **Training Dataset**

- A training dataset is a dataset of examples used for learning, that is to fit the parameters (e.g., weights) of, for example, a classifier
- Most approaches that search through training data for empirical relationships tend to overfit the data, meaning that they can identify and exploit apparent relationships in the training data that do not hold in general

**Source: Wikipedia**

# *Training, Validation and Testing*

## **Validation Dataset**

- A validation dataset is a dataset of examples used to tune the Hyperparameter (i.e. the architecture) of a classifier
- It is sometimes also called the development set or the "dev set"
- In artificial neural networks, a hyperparameter is, for example, the number of hidden units.
- It, as well as the testing set (as mentioned above), should follow the same probability distribution as the training dataset.
- In order to avoid overfitting, when any classification parameter needs to be adjusted, it is necessary to have a validation dataset in addition to the training and test datasets.
- For example, if the most suitable classifier for the problem is sought, the training dataset is used to train the candidate algorithms, the validation dataset is used to compare their performances and decide which one to take and, finally, the test dataset is used to obtain the performance characteristics such as accuracy, sensitivity, specificity, F-measure, and so on.
- **The validation dataset functions as a hybrid: it is training data used by testing, but neither as part of the low-level training nor as part of the final testing**

# *Training, Validation and Testing*

## **Basic Process**

- Since our goal is to find the network having the best performance on new data, the simplest approach to the **comparison of different networks** is to evaluate the **error function using data which is independent of that used for training**
- Various networks are trained by minimization of an appropriate error function defined with respect to a training data set.
- The performance of the networks is then compared by evaluating the error function using an independent validation set, and the network having the smallest error with respect to the validation set is selected
- This approach is called the **hold out method**
- Since this procedure can itself lead to some overfitting to the validation set, the performance of the selected network should be confirmed by measuring its performance on a third independent set of data called a test set.

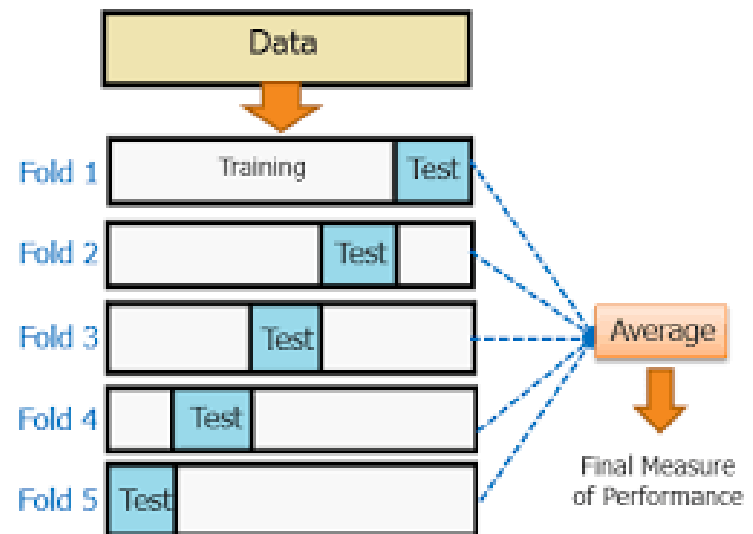
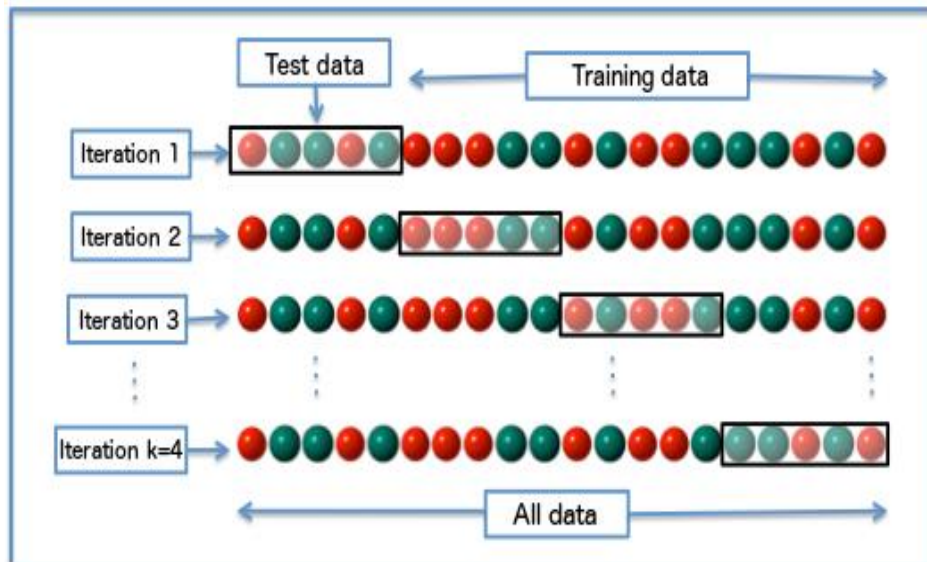
# Cross Validation

- A dataset can be repeatedly split into a training dataset and a validation dataset: this is known as cross-validation.
- Cross-validation doesn't work in situations where you can't shuffle your data, most notably in time-series

# Cross Validation

## K Folds Cross Validation Method

1. Divide the sample data into k parts.
2. Use k-1 of the parts for training, and 1 for testing.
3. Repeat the procedure k times, rotating the test set.
4. Determine an expected performance metric (mean square error, misclassification error rate, confidence interval, or other appropriate metric) based on the results across the iterations



# ***Model Evaluation and Selection***

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier's accuracy:
  - Holdout method, random subsampling
  - Cross-validation
  - Bootstrap
- Comparing classifiers:
  - Confidence intervals
  - Cost-benefit analysis and ROC Curves

# Classifier Evaluation Metrics: Confusion Matrix

## Confusion Matrix:

Actual class\Predicted class	$C_1$	$\neg C_1$
$C_1$	True Positives (TP)	False Negatives (FN)
$\neg C_1$	False Positives (FP)	True Negatives (TN)

## Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	6954	46	7000
buy_computer = no	412	2588	3000
Total	7366	2634	10000

- Given  $m$  classes, an entry,  $\mathbf{CM}_{i,j}$  in a **confusion matrix** indicates # of tuples in class  $i$  that were labeled by the classifier as class  $j$
- May have extra rows/columns to provide totals

# Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	TP	FN	P
¬C	FP	TN	N
	P'	N'	All

- **Classifier Accuracy**, or recognition rate: percentage of test set tuples that are correctly classified

$$\text{Accuracy} = (TP + TN)/All$$

- **Error rate**:  $1 - \text{accuracy}$ , or  
 $\text{Error rate} = (FP + FN)/All$

- **Class Imbalance Problem:**

- One class may be *rare*, e.g. fraud, or HIV-positive
- Significant *majority of the negative class* and minority of the positive class
- **Sensitivity**: True Positive recognition rate
  - **Sensitivity** =  $TP/P$
- **Specificity**: True Negative recognition rate
  - **Specificity** =  $TN/N$

# Classifier Evaluation Metrics:

## Precision and Recall, and F-measures

- **Precision:** exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

$$recall = \frac{TP}{TP + FN}$$

- Perfect score is 1.0
- Inverse relationship between precision & recall
- **F measure ( $F_1$  or F-score):** harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- **$F_\beta$ :** weighted measure of precision and recall
  - assigns  $\beta$  times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

# Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	<b>90</b>	<b>210</b>	300	30.00 ( <i>sensitivity</i> )
cancer = no	<b>140</b>	<b>9560</b>	9700	98.56 ( <i>specificity</i> )
Total	230	9770	10000	96.40 ( <i>accuracy</i> )

–  $Precision = 90/230 = 39.13\%$

$Recall = 90/300 = 30.00\%$

# Confusion Matrix

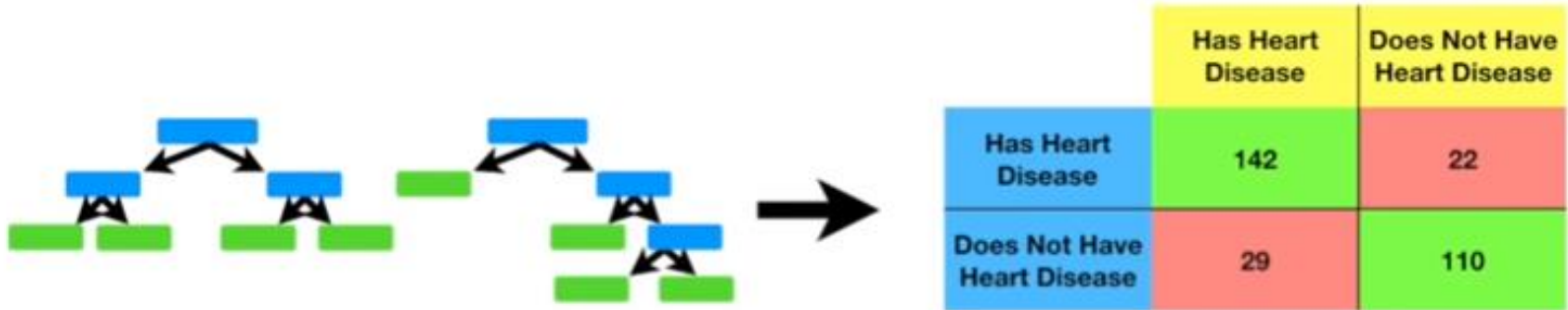
Actual	Predicted		
		Negative	Positive
	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

- Can be used to compare performance of an algorithm or compare performances of different ML Algorithms

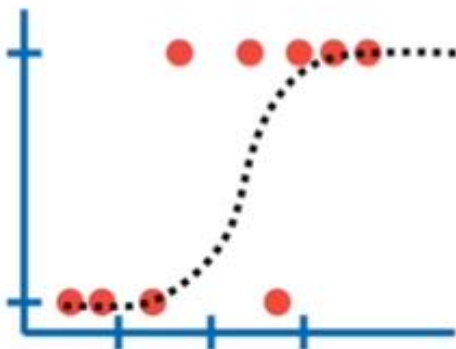
	Thing 1	Thing 2	Thing 3	Thing 4
Thing 1				
Thing 2				
Thing 3				
Thing 4				

confusion matrix with 4 rows and 4 columns...

# Confusion Matrix



These two **Confusion Matrices** are very similar and make it hard to choose which machine learning method is a better fit for this data.



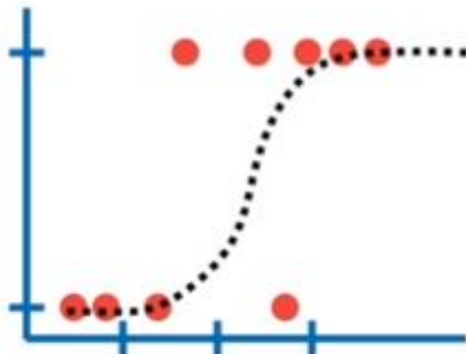
	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	139	20
Does Not Have Heart Disease	32	112

# Confusion Matrix



	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	142	22
Does Not Have Heart Disease	29	110

We'll talk about more sophisticated metrics, like **Sensitivity, Specificity, ROC** and **AUC**, that can help us make a decision



	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	139	20
Does Not Have Heart Disease	32	112

# Sensitivity and Specificity

**True Positives** are patients that *had heart disease* that were also predicted to have heart disease.

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	
	Does Not Have Heart Disease		

# Sensitivity and Specificity

**True Negatives** are patients that *did not have heart disease* and were predicted not to have heart disease.

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	
	Does Not Have Heart Disease		True Negatives

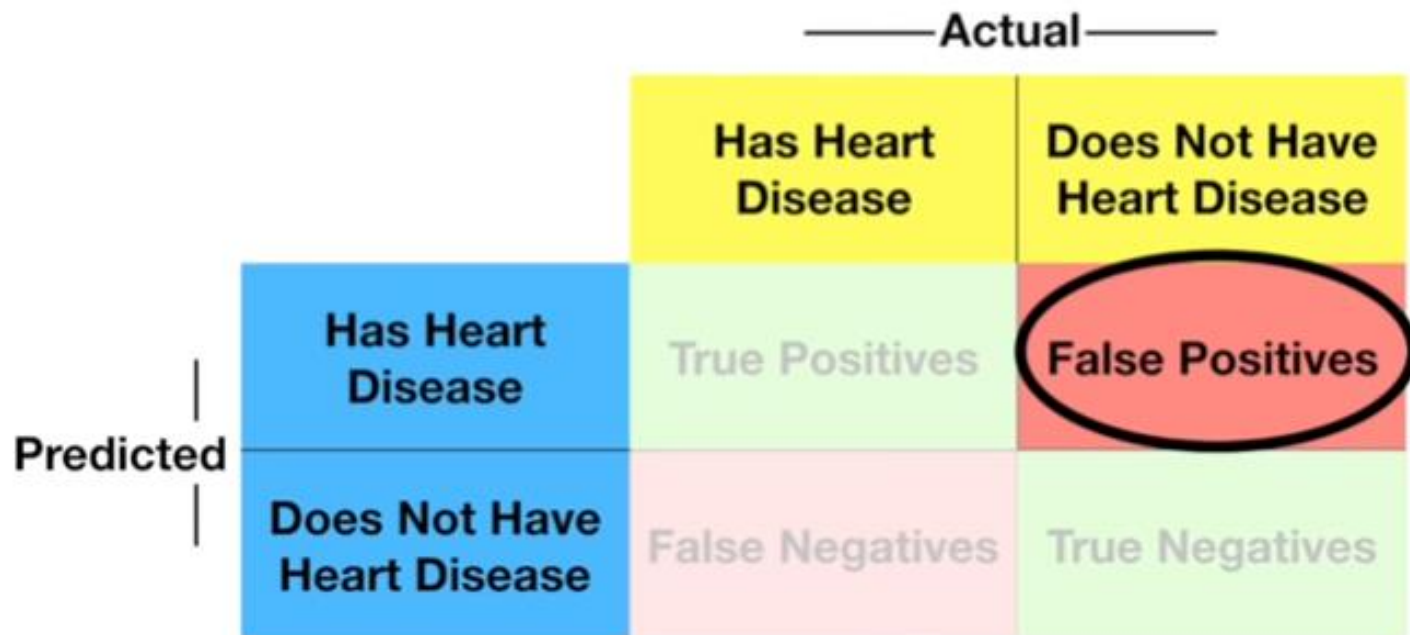
# Sensitivity and Specificity

**False Negatives** are when a patient has heart disease, but the prediction said they didn't.

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	
	Does Not Have Heart Disease	False Negatives	True Negatives

# Sensitivity and Specificity

**False Positives** are patients that do not have heart disease, but the prediction says that they do.



A confusion matrix for heart disease prediction. The columns represent the 'Actual' status (Has Heart Disease, Does Not Have Heart Disease) and the rows represent the 'Predicted' status (Has Heart Disease, Does Not Have Heart Disease). The 'False Positives' cell, representing patients predicted to have heart disease but who do not, is circled in red. An arrow points from the text 'False Positives are patients that do not have heart disease, but the prediction says that they do.' to this circled cell.

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	<b>False Positives</b>
	Does Not Have Heart Disease	False Negatives	True Negatives

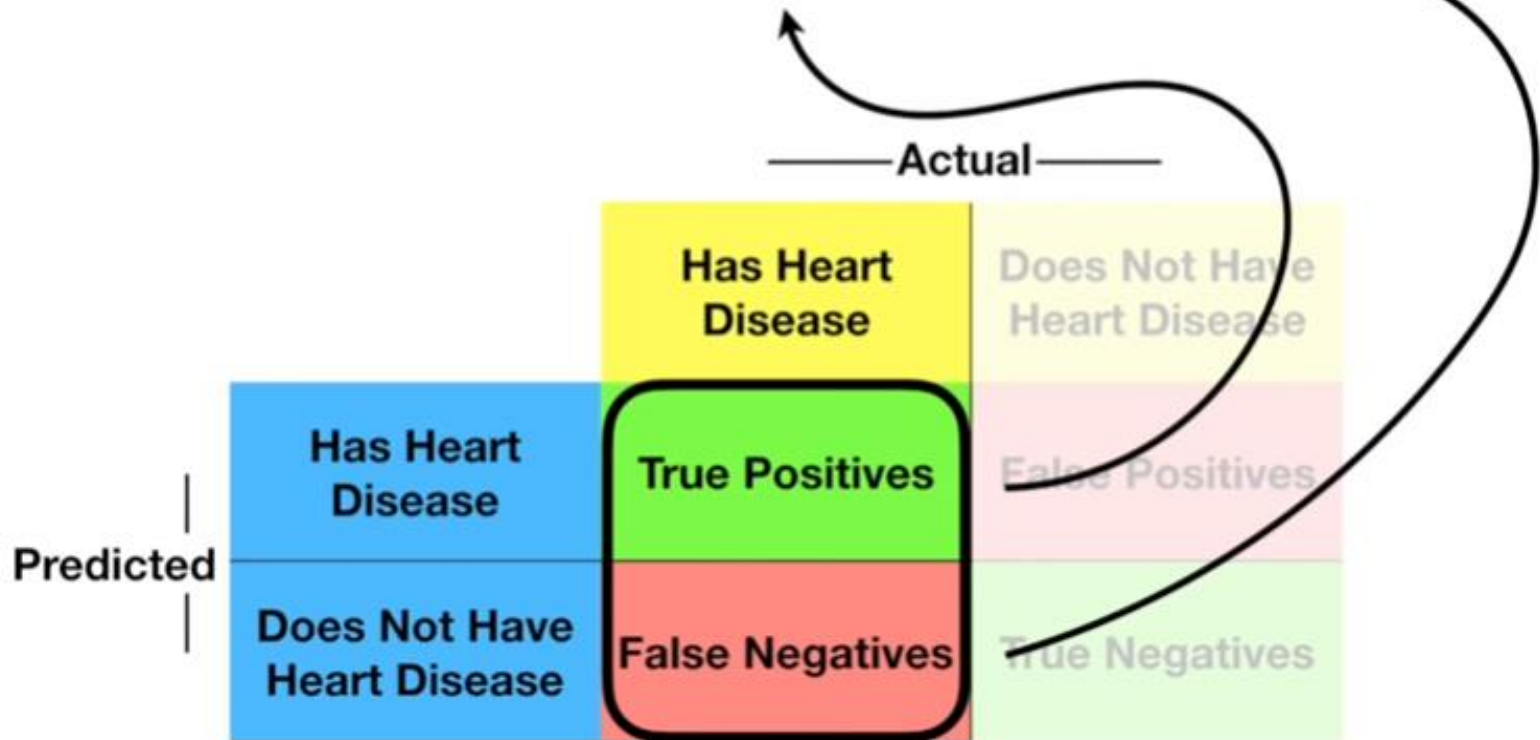
# Sensitivity and Specificity

In this case, **Sensitivity** tells us what percentage of patients *with* heart disease were correctly identified.

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	False Positives
	Does Not Have Heart Disease	False Negatives	True Negatives

# Sensitivity and Specificity

$$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$



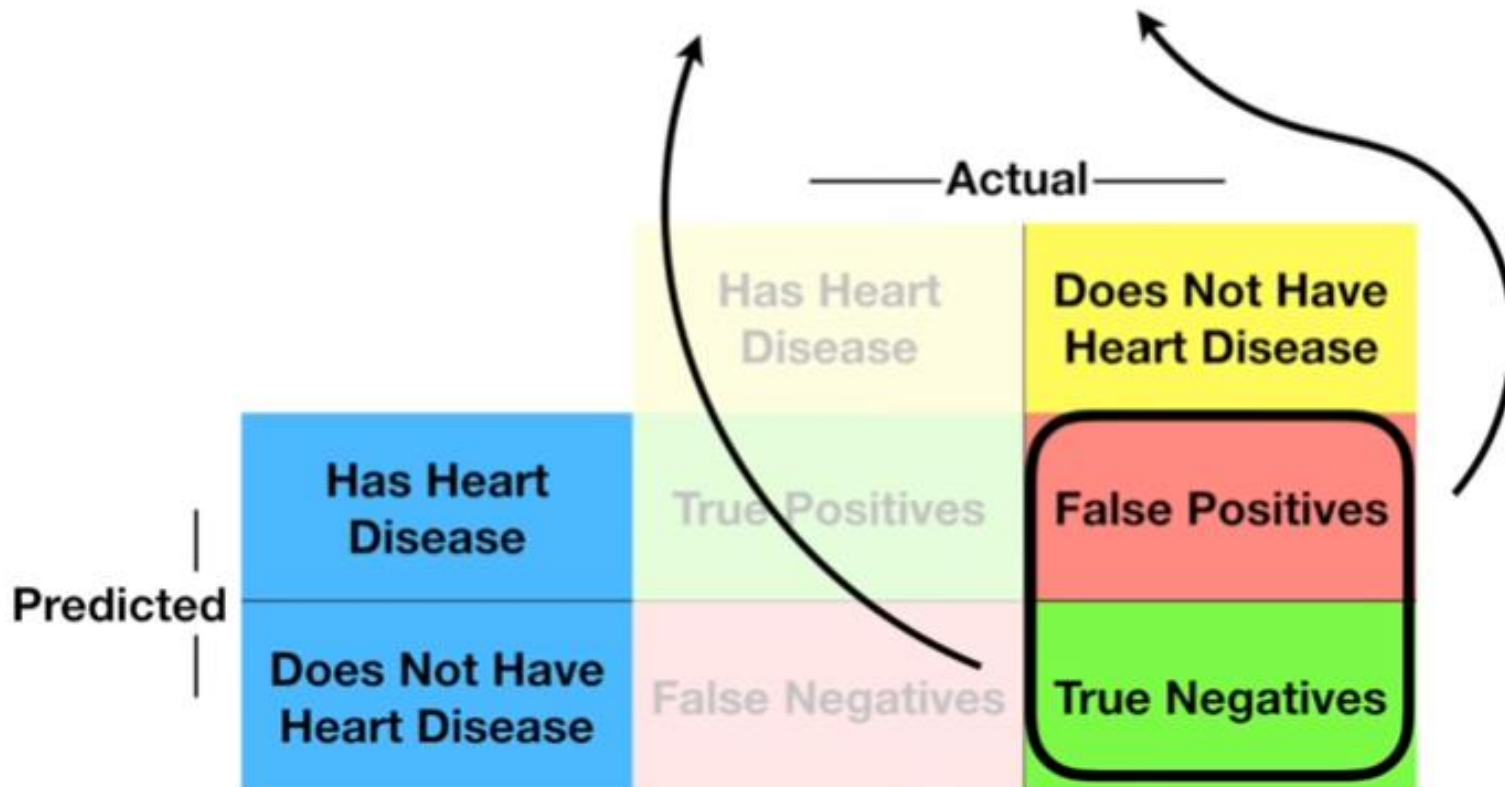
# Sensitivity and Specificity

**Specificity** tells us what percentage of patients *without* heart disease were correctly identified.

		———Actual———	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	False Positives
	Does Not Have Heart Disease	False Negatives	True Negatives

# Sensitivity and Specificity

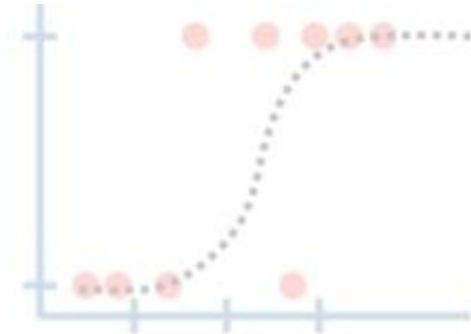
$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$



# Sensitivity and Specificity

$$\text{Sensitivity} = \frac{139}{139 + 32} = 0.81$$

**Sensitivity** tells us that **81%** of the people *with* Heart Disease were correctly identified by the **Logistic Regression** model.

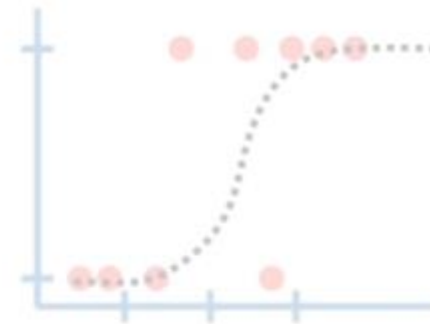


	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	139	20
Does Not Have Heart Disease	32	112

# Sensitivity and Specificity

$$\text{Specificity} = \frac{112}{112 + 20} = 0.85$$

**Specificity** tells us that **85%** of the people *without* Heart Disease were correctly identified by the **Logistic Regression** model.



	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	139	20
Does Not Have Heart Disease	32	112

# Sensitivity and Specificity

$$\text{Sensitivity} = \frac{142}{142 + 29} = 0.83$$

$$\text{Specificity} = \frac{110}{110 + 22} = 0.83$$



		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	142	22
	Does Not Have Heart Disease	29	110

# Sensitivity and Specificity



**Sensitivity** = 0.81

**Specificity** = 0.85

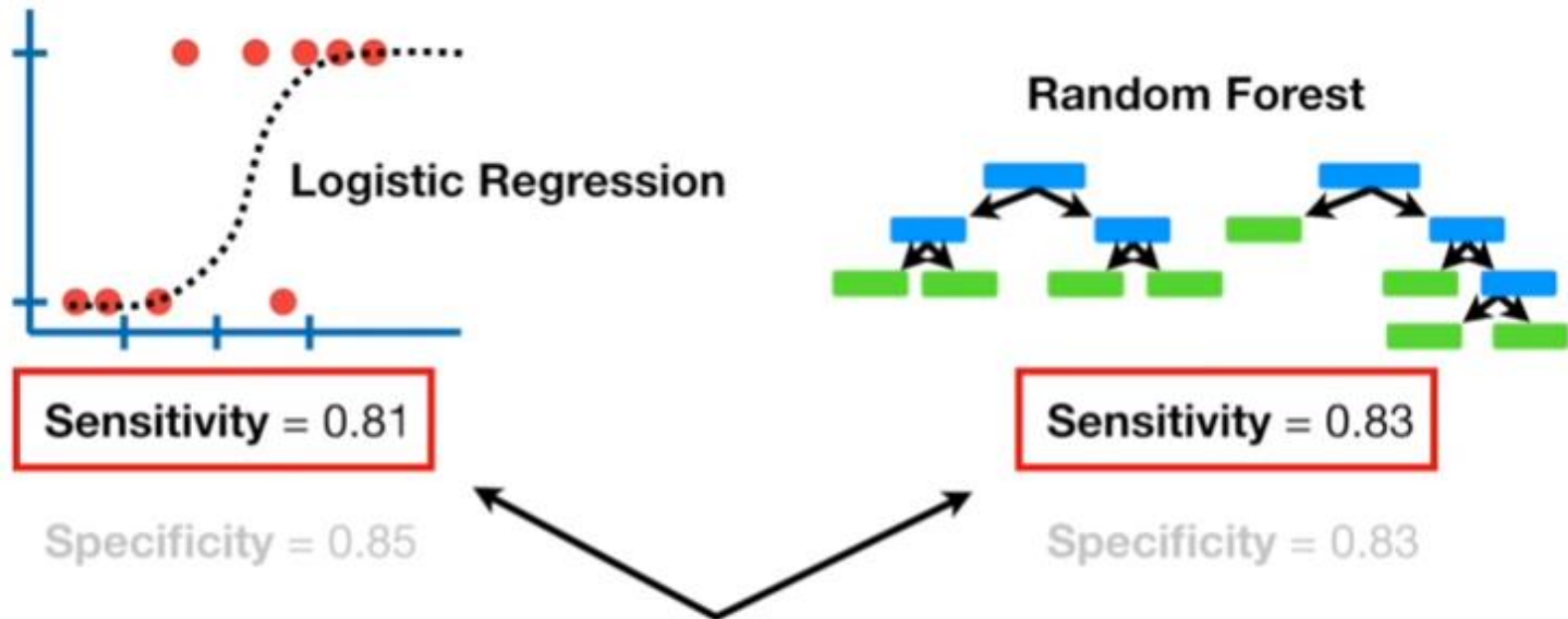


**Sensitivity** = 0.83

**Specificity** = 0.83

...to the values we calculated for the  
**Random Forest.**

# Sensitivity and Specificity



**Sensitivity** tells us that the **Random Forest** is slightly better at correctly identifying *positives*, which, in this case, are patients *with* heart disease.

# Sensitivity and Specificity



Sensitivity = 0.81

**Specificity = 0.85**



Sensitivity = 0.83

**Specificity = 0.83**

**Specificity** tells us that **Logistic Regression** is slightly better at correctly identifying *negatives*, which, in this case, are patients *without* heart disease.

# Sensitivity and Specificity



**Sensitivity** = 0.81

**Specificity** = 0.85

We would choose the **Logistic Regression** model if correctly identifying patients **without** heart disease was more important than correctly identifying patients **with** heart disease.



**Sensitivity** = 0.83

**Specificity** = 0.83

# Sensitivity and Specificity



Sensitivity = 0.81

Specificity = 0.85



Sensitivity = 0.83

Specificity = 0.83

Alternatively, we would choose the **Random Forest** model if correctly identifying patients **with** heart disease was more important than correctly identifying patients **without** heart disease.

# Sensitivity and Specificity

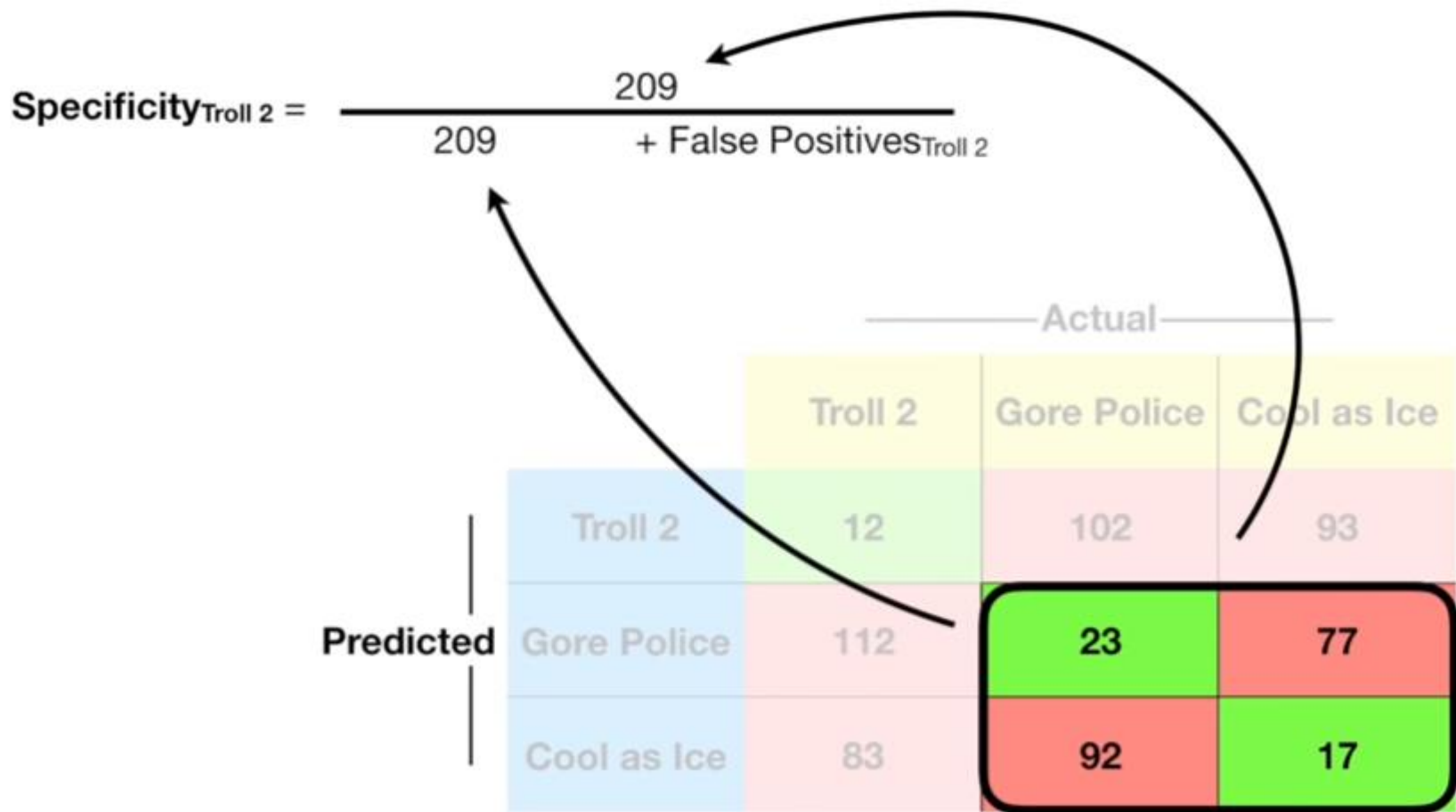
$$\text{Sensitivity}_{\text{Troll 2}} = \frac{12}{12 + 195}$$

$$\text{Sensitivity}_{\text{Troll 2}} = \frac{12}{12 + 195} = 0.11$$

**Sensitivity**<sub>Troll 2</sub> tells us that only 11% of the people that loved the movie **Troll 2** more than **Gore Police** or **Cool as Ice** were correctly identified.

		Actual		
		Troll 2	Gore Police	Cool as Ice
Predicted	Troll 2	12	102	93
	Gore Police	112	23	77
	Cool as Ice	83	92	17

# Sensitivity and Specificity



# Sensitivity and Specificity

$$\text{Specificity}_{\text{Troll 2}} = \frac{209}{209 + 195}$$

$$\text{Specificity}_{\text{Troll 2}} = \frac{209}{209 + 195} = 0.52$$

**Specificity**<sub>Troll 2</sub> tells us that **52%** of the people who loved **Gore Police** or **Cool as Ice** more than **Troll 2** were correctly identified.

		Actual		
		Troll 2	Gore Police	Cool as Ice
Predicted	Troll 2	12	102	93
	Gore Police	112	23	77
	Cool as Ice	83	92	17