CIVL 7012/8012

Introduction to Machine Learning and basic algorithms

What is Machine Learning?

- Field of study that enables computers to learn by itself without being programmed.
- "Learning" implies improving on certain task T, with respect to performance metric P based on training experience E

Some examples

- Identifying spams in email
  T: Classify emails as legitimate or spam
  P: Percentage of accurate classifications
  E: Email database

- Detecting damages in structures
  T: Detect the location and extent of damages in structure
  P: Percentages of accurate detections
  E: Visual and sensory data

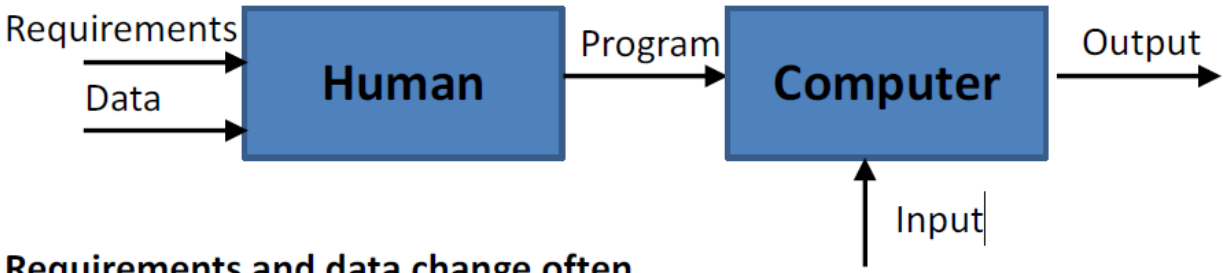
Some more examples

- Classifying soil using photographs
  T: Classifying soil class and type
  P: Percentages of accurate classifications
  E: Visual data such as photographs

- Identifying spoken words (e.g. Apple Siri, Amazon Alexa, etc.)
  T: Classify and identify words spoken by someone
  P: Percentage of accurate identifications
  E: Database of words spoken by people with different accents

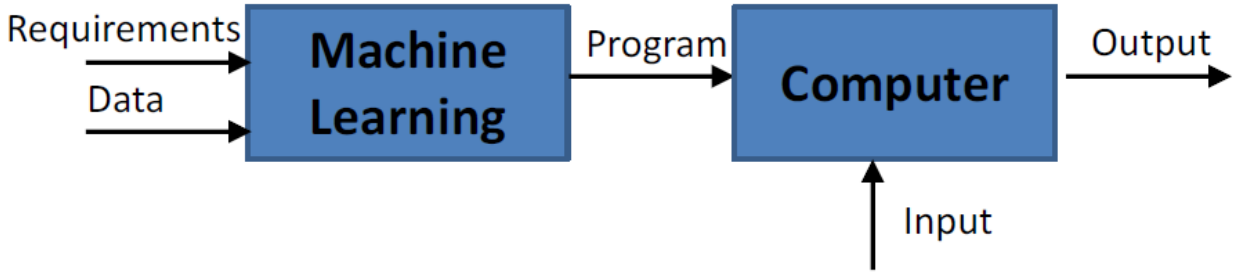
How is ML different from traditional modeling?

Traditional modeling



Requirements and data change often

Machine Learning



For ML algorithms the data is enough for the computer to recognize patterns and make predictions

ML Terminology

- Generalization: The ability of ML algorithm to fit new data.
- Features, class and examples:

  See figure below for a sample data with dependent and independent variables from HW 4 Q3.

Features (Independent variables)

Class label or target class (Dependent variable)

Examples or samples (Observations)

Speed	Route	Seatbelt	Age	Income	Marital status
Low	1	1	1	2	Married
Medium	2	0	3	3	Married
High	1	0	1	1	Unmarried
High	1	1	2	2	Married

Types of learning

1. **Supervised learning**

- Used when the data has desired outputs
- E.g. regression, decision trees

2. **Unsupervised learning**

- Used when the data does not have desired outputs
- E.g. Clustering algorithm such as K-means

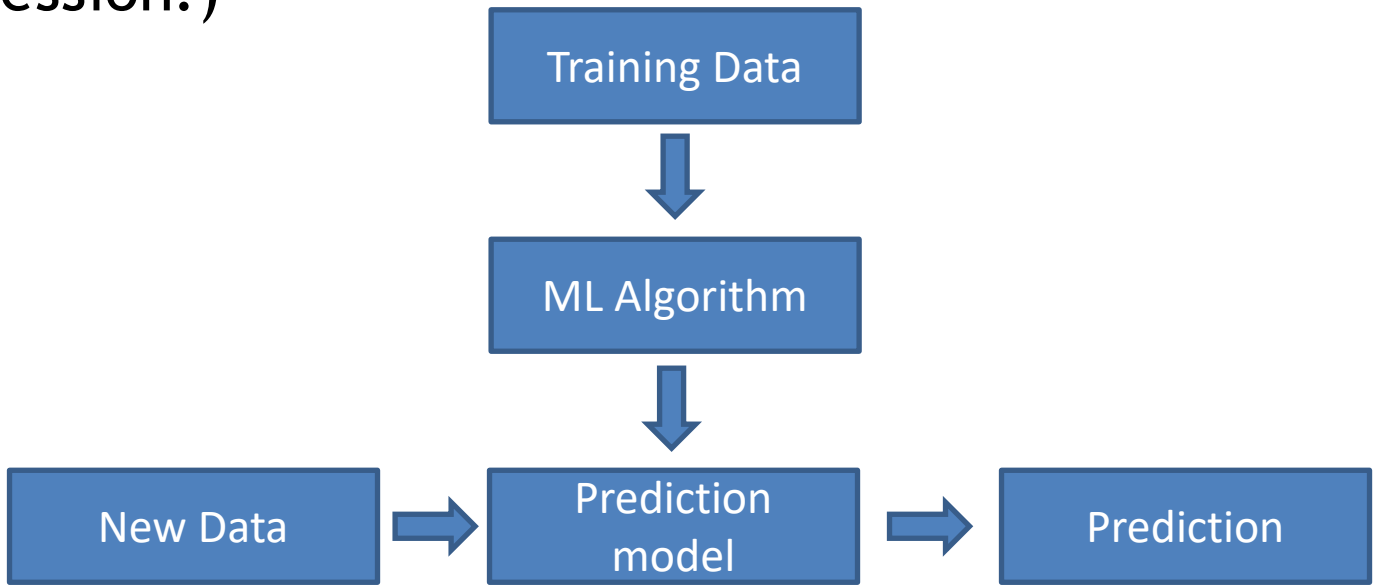
3. **Reinforcement learning**

- Commonly used in robotics
- Learning based on rewards gained from actions

Types of learning

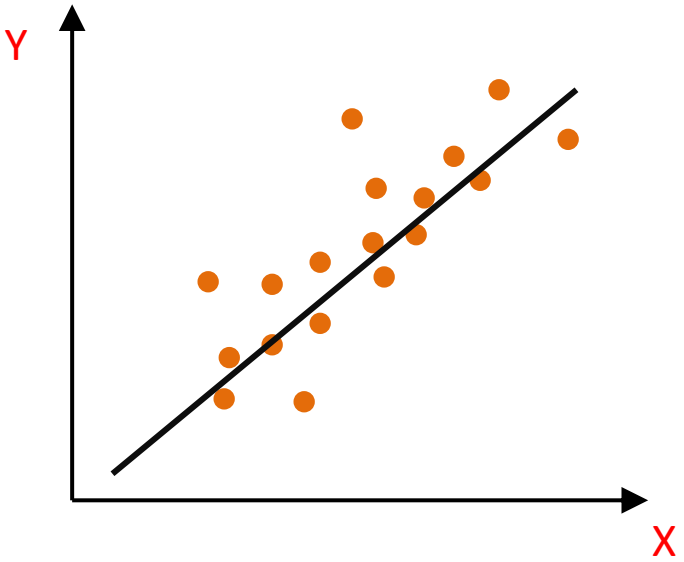
1. Supervised learning

- Train a model using training data
- Predict on a new unseen data
- Training data includes desired outputs (Remember regression?)

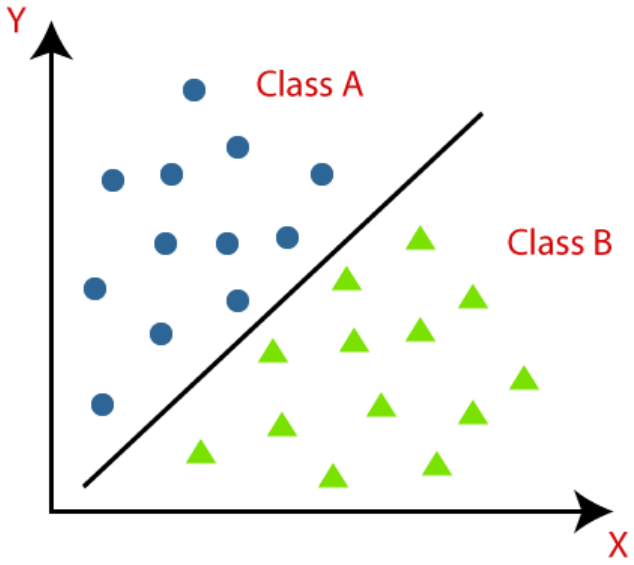


Supervised learning

- For a continuous data prediction-regression
- For categorical prediction-classification (e.g. binary logit)



Regression



Classification

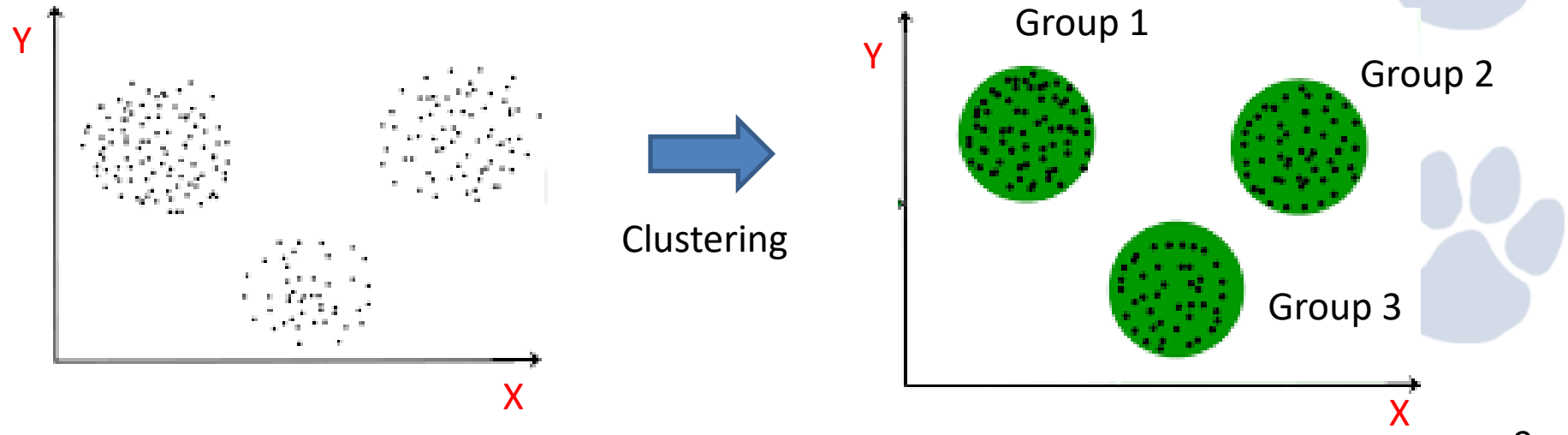
Image source: https://www.javatpoint.com/classification-algorithm-in-machine-learning

Types of learning

2. Unsupervised learning
- Training data does not include desired outputs
- Creates ‘clusters’ of similar data

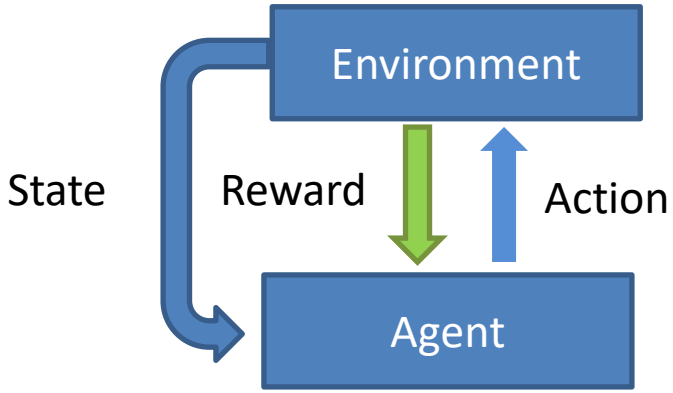
Example.

Figure below shows clustering of unlabeled data into three groups based on their similarity with features X, and Y



Types of learning

3. Reinforcement learning
- Considerations in reinforcement learning
  - There are finite number of *states* with finite number of *actions*
  - The *agent* improves its's performance based on *reward* from *actions* or interaction with the *environment*.



Example 1. A chess engine (*agent*) training on chess moves will base its moves on the state of the chess board (*environment*) with the reward positive if win, negative if lose.

Reinforcement learning

Example 2

Consider a self driving car stopped at a red-light signal. The lights turn green and the car starts moving. Here the agent is the car and it is moving from one state to another. This actions produces some reward.

On the contrary if the car does not start and remains at the intersection despite green signal, it receives a traffic ticket. This is negative reward.

Some common learners

- Naïve Bayes (Supervised)
- Decision trees (Supervised)
- Perceptron and Artificial Neural Networks (Supervised/unsupervised)
- K-means clustering (Unsupervised)

Naïve Bayes

- Naïve Bayes is based on Bayes theorem for conditional probability:

$$P(y_i|X) = \frac{P(X|y_i)P(y_i)}{P(X)}$$

- As we are calculating probabilities for all classes the above equation can be written as

$$P(y_i|X) \propto P(X|y_i)P(y_i)$$

Naïve Bayes

- Even with this, it becomes difficult to learn $P(X|y_i)$

Example

- Consider *X* has only *2* features: gender(male, female) and marital status (married and unmarried), and there are two possible outcomes for target class wage(low and high).
- To apply Bayes theorem we will need to learn 2^n conditional probabilities.

Naïve Bayes

- With large number of features Bayes theorem becomes difficult to apply
- To learn $P(X|y_i)$ where *X* has *n* features, we need to learn the conditional probabilities $P(x_1|y_i), P(x_2|y_i) \ldots \ldots P(x_n|y_i)$ which is much easier
- We make a naïve assumption to simplify problems i.e. features are conditionally independent given class.

Naïve Bayes

$$P(x_1, x_2|y) = P(x_1|y)P(x_2|y)$$

OR

$$P(x_1, x_2 \ldots . x_n|y) = \prod_i P(x_i|y)$$

- Classification is then based on class selection with largest probability

$$y^* = argmax_y P(y)^* \prod_i P(x_i|y)$$

Naïve Bayes

Example. For the given features W,X and Y and Class label, predict the class label for W=F, X=T, Y=F.

W	X	Y	Class
T	T	T	T
T	F	T	F
T	F	F	F
F	T	T	F
F	F	F	T
F	T	F	???

Naïve Bayes

We calculate conditional probabilities for the outcomes and choose the one that is most likely:

- $P(W = F, X = T, Y = F | Class = T) =$

$P(W = F|Class = T) * P(X = T|Class = T)^* P(Y = F|Class = T) =$
$0.5 * 0.5 * 0.5 = 0.125$

$P(Class = T|W = F, X = T, Y = F) = 0.125 * P(Class = T)$
$= 0.125 * 0.4 = 0.05$

Similarly;

- $P = (Class = F|W = F, X = T, Y = F) =$

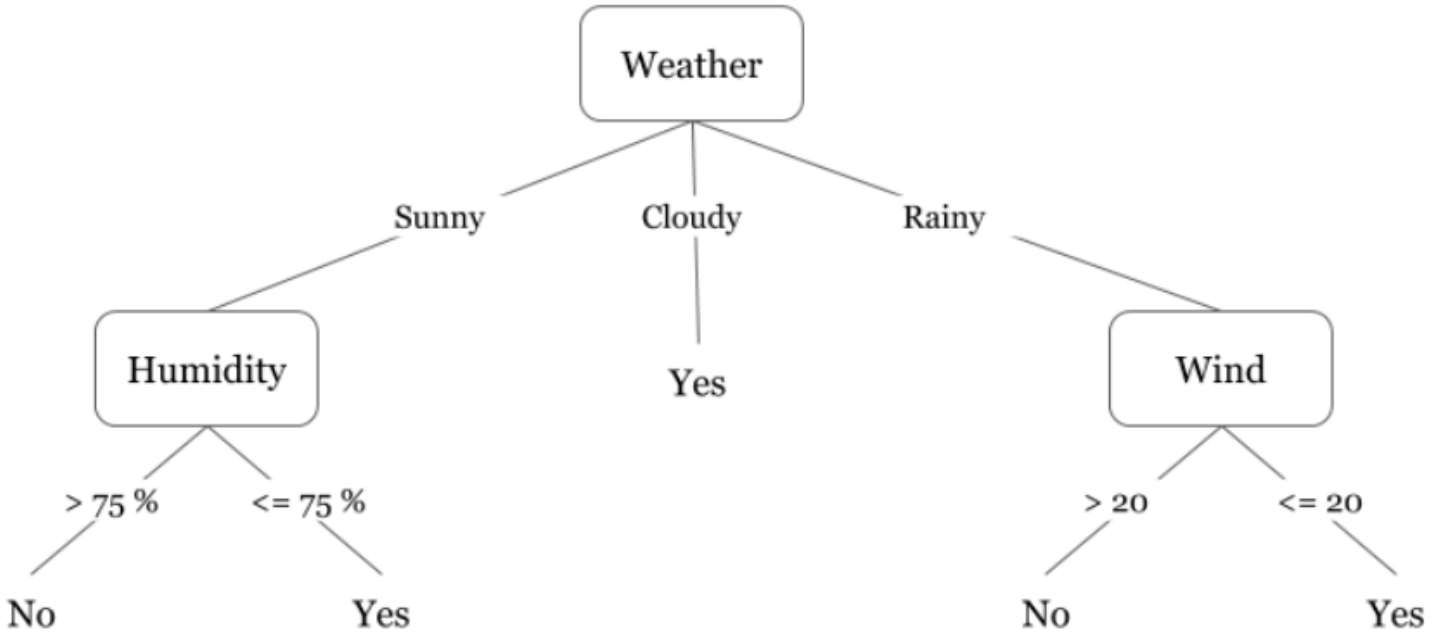
$P(W = F|Class = F) * P(X = T|Class = F)^* P(Y = F|Class = F) *$
$P(Class = F) = 0.022$

- The predicted class is therefore T

Decision trees

- Uses divide and conquer approach to split data to smaller subsets
- More intuitively it is like a series of IF-THEN statements

Example. Consider decision tree for playing badminton based on conditions outside



Decision trees

- The root node represents the entire dataset.
- The algorithm chooses a feature that is most predictive of the target class and then partitions the examples into groups based on values of the feature. Here, “sunny”, “cloudy”, and “rainy”.
- The algorithm continues this divide and conquer approach until there are no feature remaining or the size limit for the tree is reached.

Decision trees

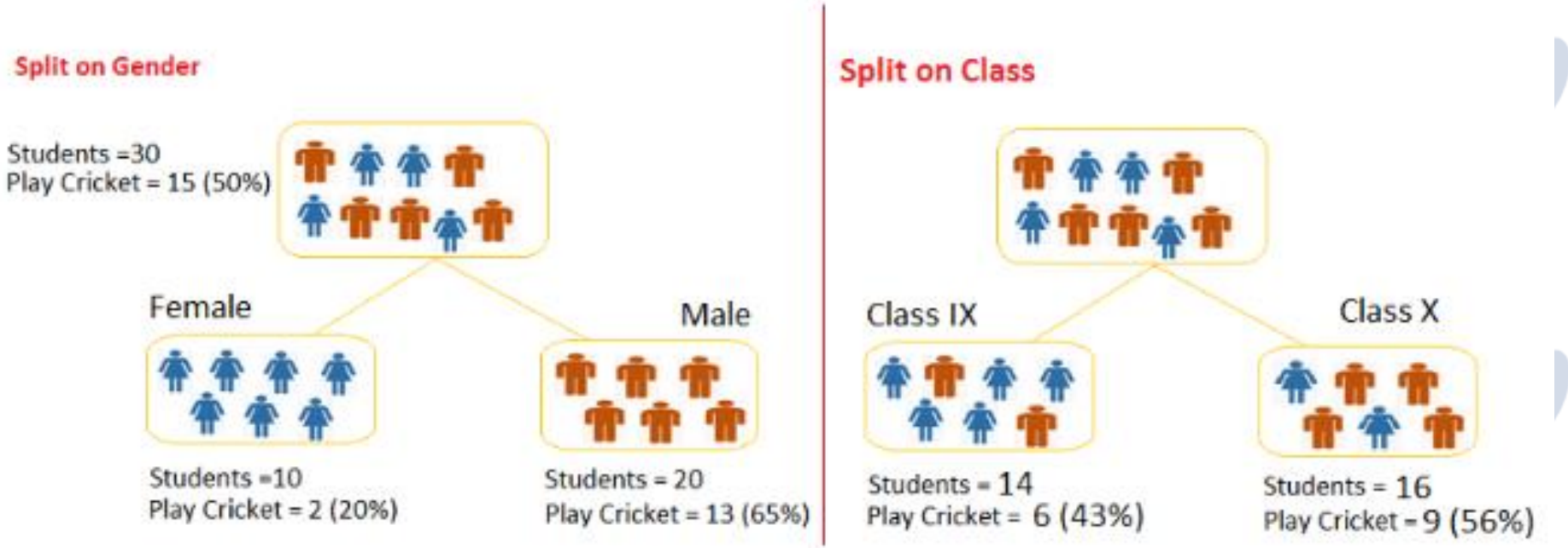
- How to decide feature for split?
- Different methods are used to decide the feature for branching.
- Some algorithms used to determine feature upon which to branch are Chi-square, entropy, and Gini index
- For example using Chi-square;
  - Calculate Chi-square for all the potential features and their target class expected on branching

$$Chi - square = \sqrt{\frac{(Target_{Actual} - Target_{Expected})^2}{Target_{Expected}}}$$

  - Branch along the feature with the highest Chi-square

Decision trees

Example of split (Taken from a Medium post by Rishabh Jain.)
We want to segregate the students based on target class (play cricket or not). There are two possible feature on which to split; 1) gender, 2) class. Select feature to be branched (split) upon.



Decision trees

Calculate Chi-square for each of the features:

Gender

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
Female	2	8	10	5	5	-3	3	1.34	1.34
Male	13	7	20	10	10	3	-3	0.95	0.95
							Total Chi-Square	4.58	

Class

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
IX	6	8	14	7	7	-1	1	0.38	0.38
X	9	7	16	8	8	1	-1	0.35	0.35
							Total Chi-Square	1.46	

Chi-square for gender is more so Gender split is more significant.

Perceptron

- Perceptron is a binary linear classifier that can separate only linearly separable data(could be a line, plane, etc.).
- Visualize perceptron decision surface as;
  - Line if for two features
  - Plane for three features
  - N-dimensional hyperplane for n features
- Given the equation for the decision surface, classification can is done (think of regression using categorical outcomes)

Perceptron

- Given inputs $x_1, x_2, \ldots\ldots x_n$, with weights w_0, w_1, $w_2 \ldots . w_n$ perceptron produces output based on a function called activation function.

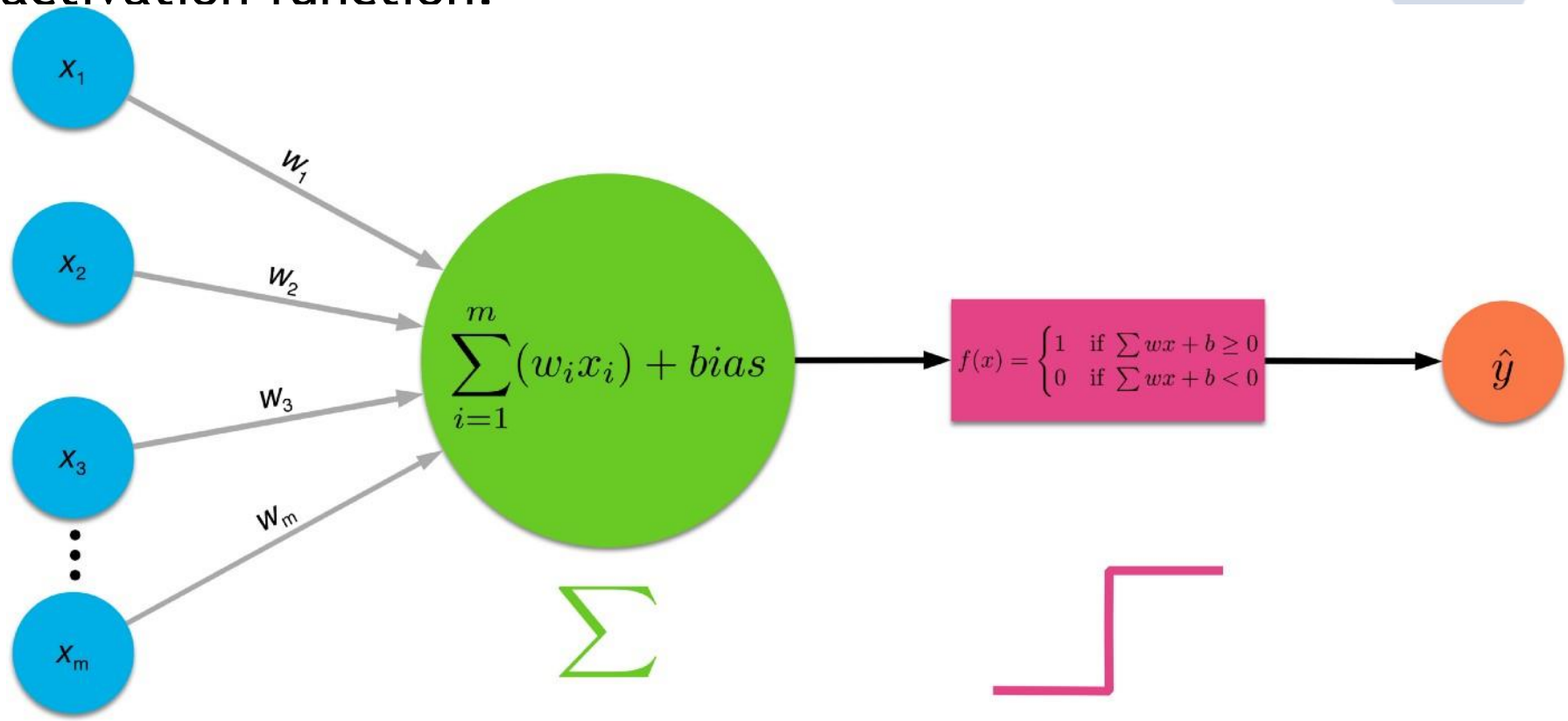


Image source: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

Perceptron

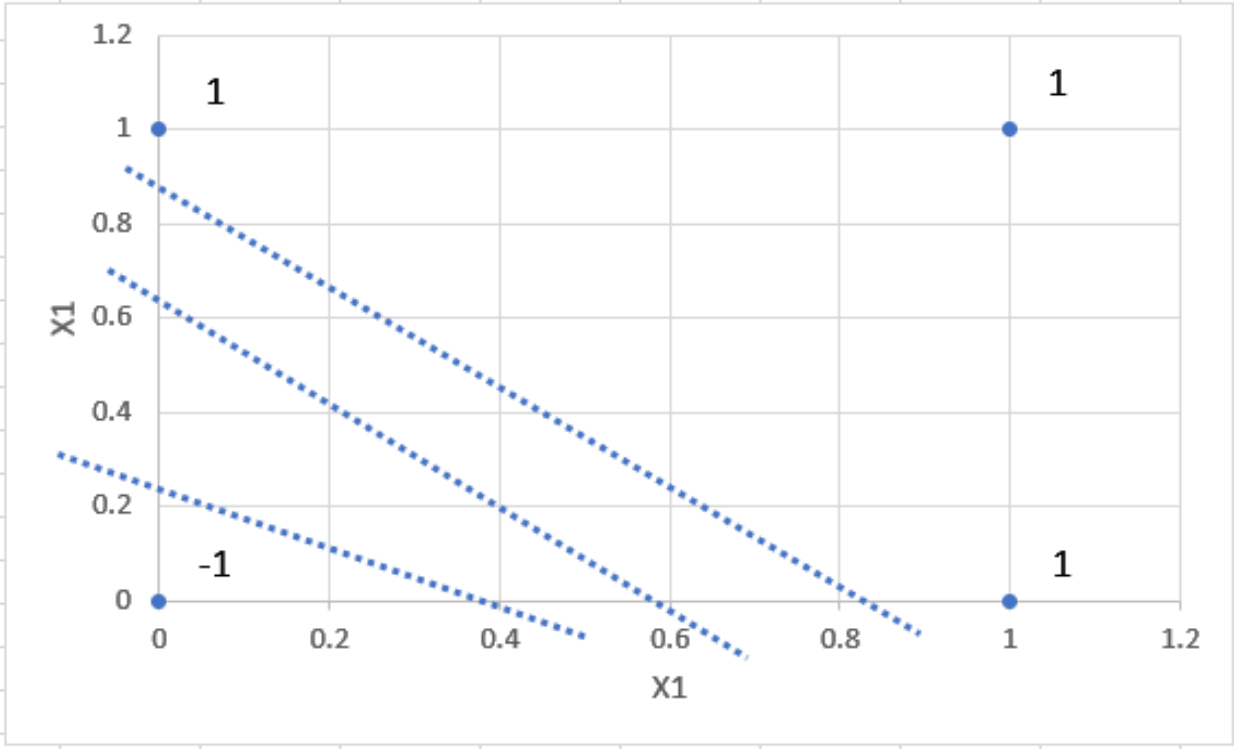
Example. Find a perceptron to classify the following data with features X1, X2 and target class Y

X1	X2	Y
0	0	-1
0	1	1
1	0	1
1	1	1

Perceptron

- The figure shows three possible perceptrons which can be represented as $w_0 + w_1 * X1 + w_2 * X2$.

Here $w_0, w_1 \; and \; w_2$ are weights that need to be estimated that can classify the points



Perceptron

- Note that infinite possible weight settings are possible.
- Let us look at one possible set of weights weight settings for the problem.
- We use the weights; $w_0 = -0.5, w_1 = 1 \; and \; w_2 = 1.$

The perceptron: $-0.5 + X1 + X2$

X1	X2	Y (True)	Y (Predicted)
0	0	-1	-0.5 (-ve so output -1)
0	1	1	0.5 (+ve so output 1)
1	0	1	0.5 (+ve so output 1)
1	1	1	1.5 (+ve so output 1)

Perceptron learning

- In the previous example; simple hit and trial with weights could help us determine the perceptron.

But What if we have 1000 features?

- Gradient descend is used to estimate the weights for the features
- Given a set of n training examples (x_1, x_2, x_3 …x_n) we compute corresponding weights (w_0, w_1, w_2, ….w_n) such that perceptron classifies the training examples correctly
- We minimize the squared difference between expected and observed output slowly by moving along the function and updating weights as we move along

Gradient descent (Taken from "Machine learning" by Tom Mitchell (1997))

Gradient Descent

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Perceptron

Let's learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Sum of $(true\ label - observed\ label)^2$ for training example, *d*

Gradient descent (Taken from "Machine learning" by Tom Mitchell (1997))

- Direction of steepest gradient on the surface is found by finding the derivative of E w.r.t. the components of vector of weights

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

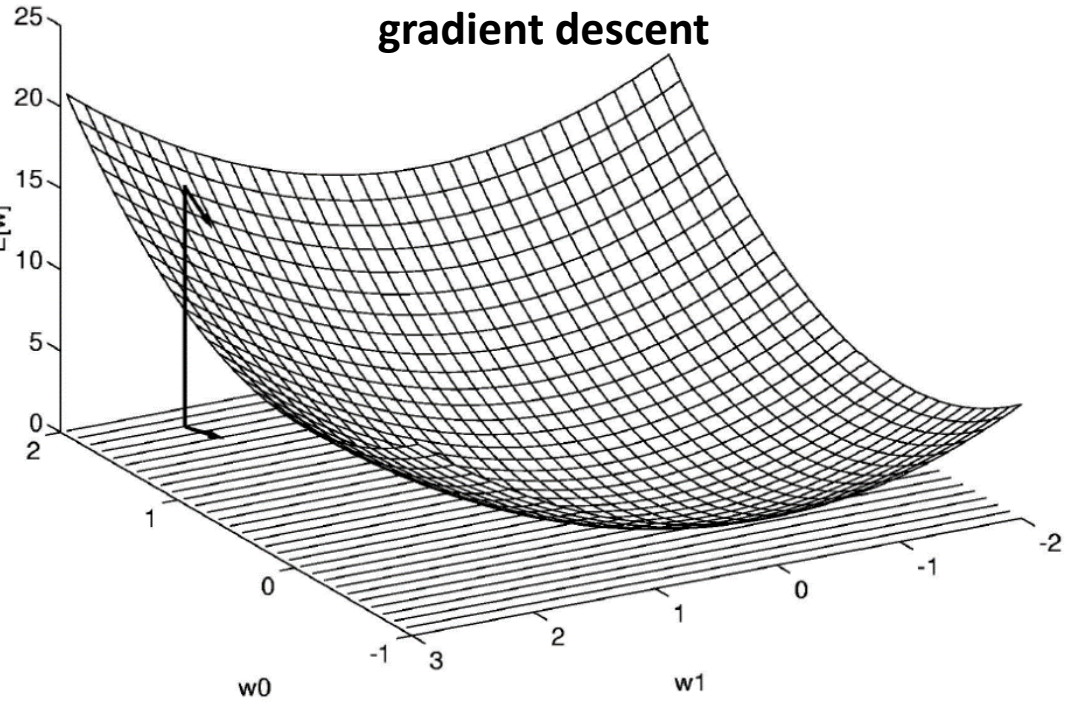
- This can be iterated over each weights by differentiation

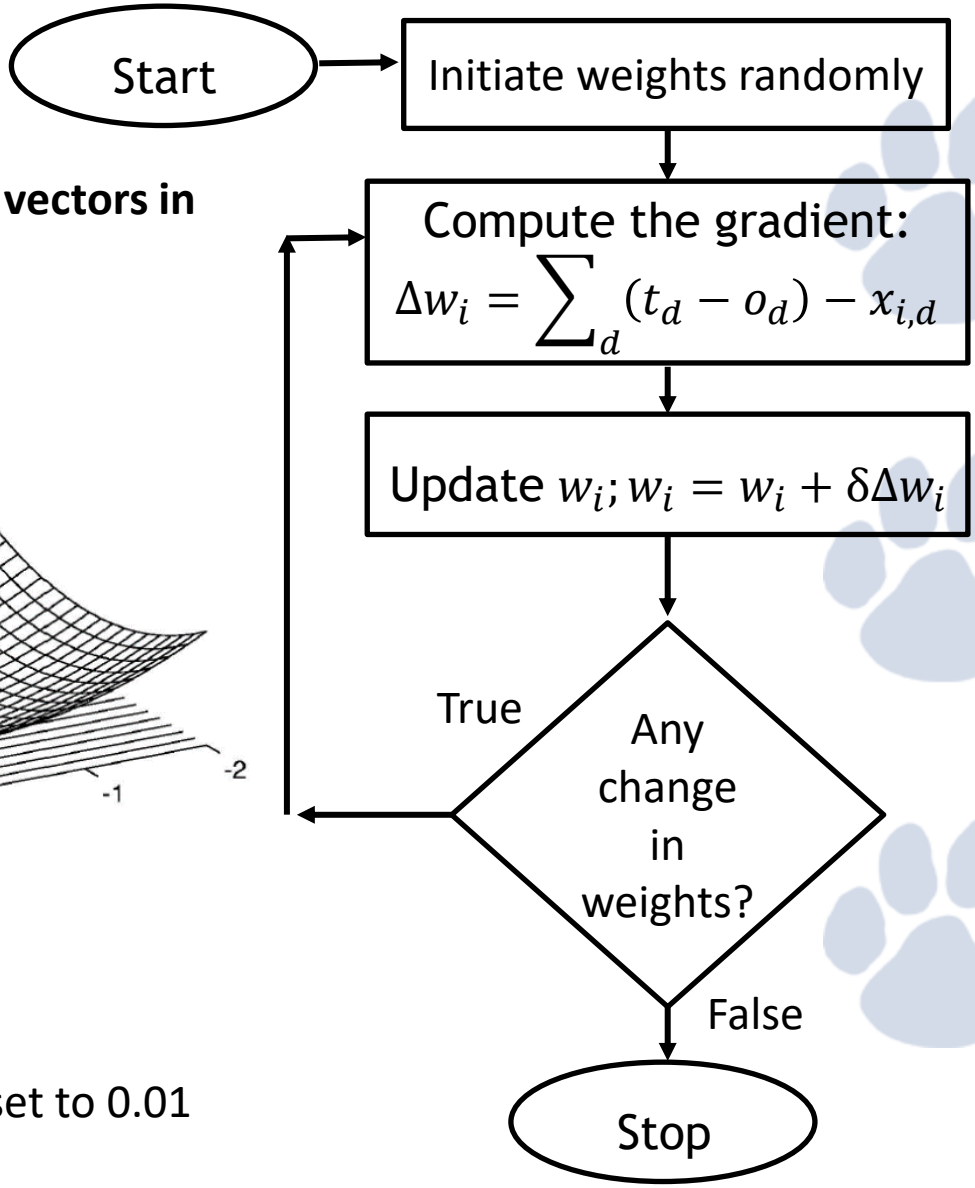
$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}$$

For training example d:
$t_d - true\ label$
$o_d - observed\ or\ predicted\ label$
$x_{i,d} - feature\ input\ i$

Gradient descend

Hypothesis space for possible weight vectors in gradient descent

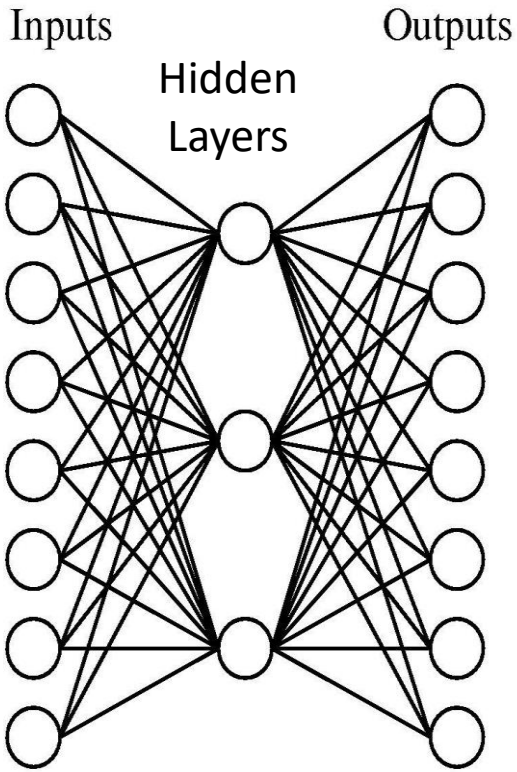




Note: δ is called learning rate typically set to 0.01

Artificial Neural Networks

- ANN is a multilayered perceptron
- There are three distinct layers; the input, hidden and output.
- Each input node represents a feature and output node represents output
- ANN are called black box because the operations in the hidden layers cannot be observed.
- Variable output produced from the layers depends on threshold functions as with perceptrons
- More complex backward and forward propagation methods are used in estimating weights by distributing errors to the nodes



K-means clustering

- Clustering clusters data in groups by grouping similar instances
- Useful in detecting patterns e.g. in
  - consumer's spending pattern
  - group emails based on keywords
- Most useful when label is not known, or we don't know what we are looking for

K-means clustering

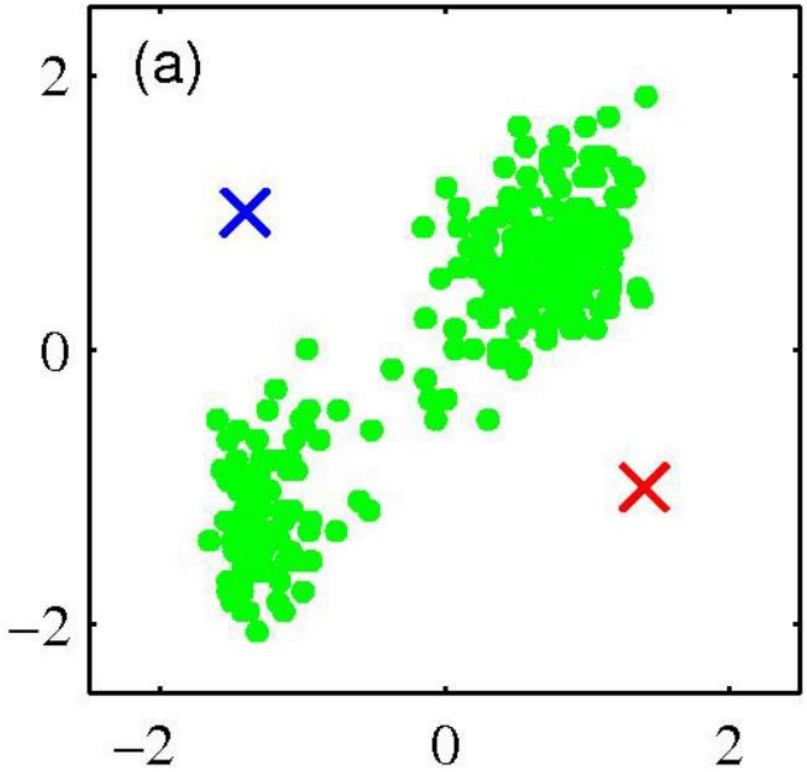
- We group similar instances together, but what does "similar" mean?
- Similarity can be based on some distance measure
- For 2D data points, grouping can be based on Euclidean distance between the points.
- Squared Euclidian distance for points $x = (x_1, x_2 \ldots x_n)$and $y = (y_1, y_2 \ldots y_n)$:

$$d(y, x) = \sum_{i}^{n} (y_i - x_i)^2$$

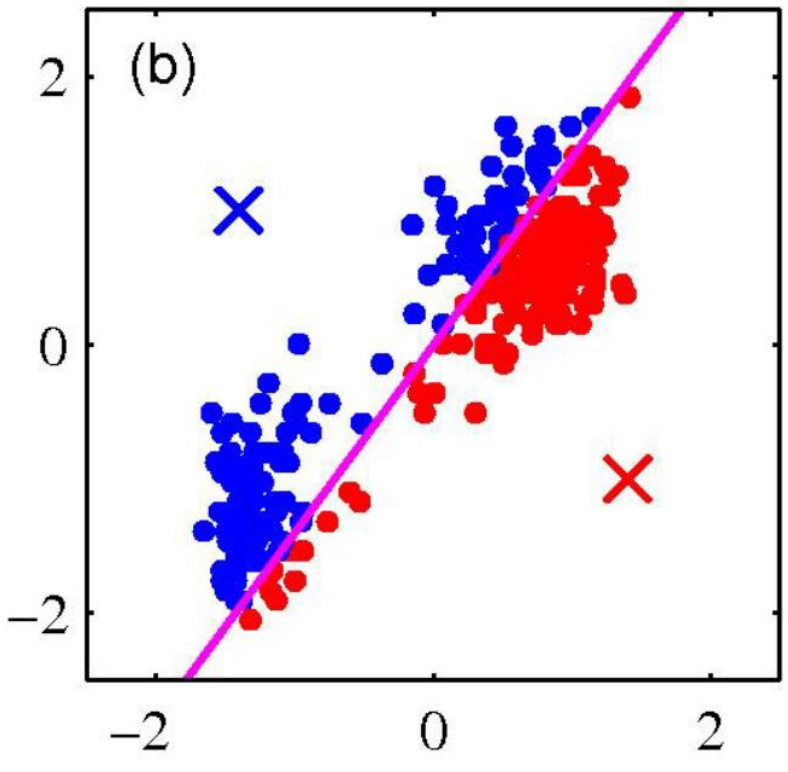
How does K-means clustering work?

- Steps in K-means clustering
  1. Pick K random points
  2. Assign data instances to the closest cluster center
  3. Update cluster center to the average of assigned points
  4. Stop when all points are assigned else go to Step 2

K-means clustering

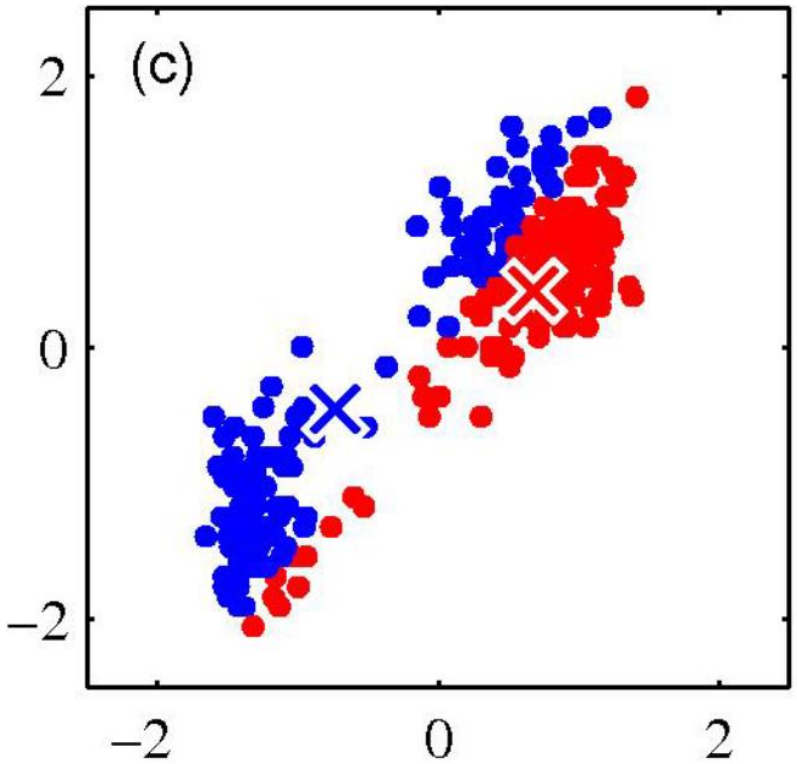


Select k=2 random points as clusters

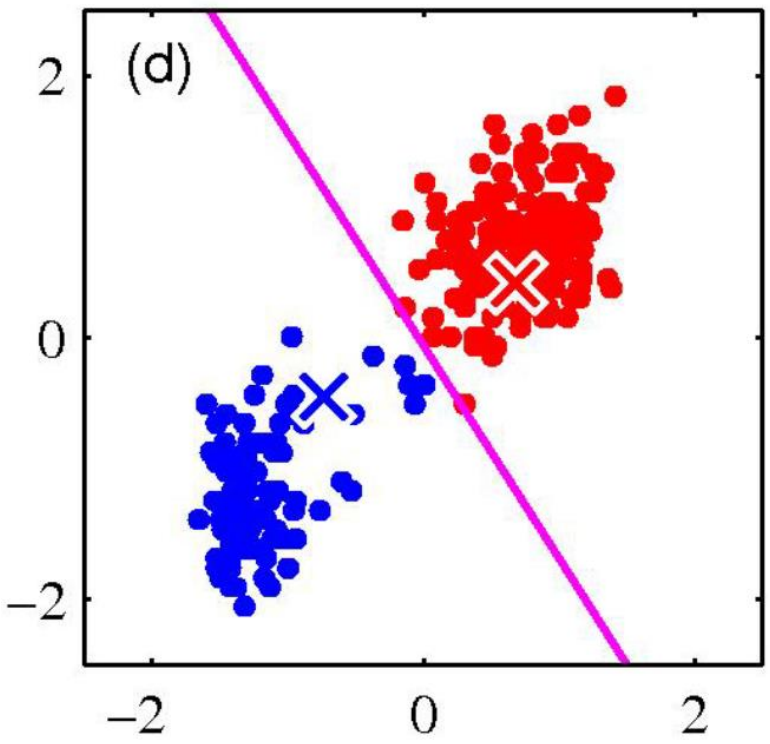


Assign data instances to the closest cluster center

K-means clustering



Update cluster center to average of the grouped points



Repeat until all instances are assigned

Evaluation metrics for ML classifiers

- Accuracy alone is not a good measure of the accuracy of an algorithm.
- Consider an algorithm for detecting email spams. Out of 10,000 emails we know there are 100 spams.
  Case 1: Any email the algorithm sees is classified as Not-spam.
  - Accuracy for spam emails only = 0/100

  Case 2: Any email the algorithm sees is classified as spam
  - Accuracy for spam = 100/100

  Is the second algorithm better?
- Cases where an outcome is less frequent is a common problem warranting use of accuracy based on rate of false positive and false negatives predictions

Precision, recall and accuracy

- Two standard measures to evaluate quality of prediction.
  - Precision: Accuracy in terms of classifying relevant instances

$$Precision = \frac{True\ positive}{Actual\ results} = \frac{True\ positive}{True\ positive + False\ positive}$$

  - Recall：Accuracy in classifying all relevant instances correctly

$$Recall = \frac{True\ positive}{Predicted\ results} = \frac{True\ positive}{True\ positive + False\ negative}$$

  - Also, $Accuracy = \frac{True\ positive + True\ negative}{Total\ instances}$

F1 score

- Combines precision and recall
- Calculated as harmonic mean of precision and recall
- F1 score, $F1 = \frac{2*Precision*Recall}{Precision+Recall}$

Example 1

Consider there are 100 known spam emails in a dataset with 10,000 emails. Find precision, recall assuming all the emails the algorithm sees are classified as Not-spam.

- True positive=0, False positive=0, false negative=100
- Precision=undefined
- Recall=0
- F1 score =undefined;
- Conclusion – not a good algorithm

Example 2

Consider there are 500 known spam emails in a dataset with 1000 emails. Find precision, recall assuming all the emails the algorithm sees are classified as spams.

- True positive=500, False positive=500, false negative=0
- Precision=0.5
- Recall=1
- F1 score =0.667;
- Conclusion – relatively good algorithm