Introduction to Machine Learning

Live Class Notes

Agenda

Part 1

- Data pre-processing
- Objective functions and evaluation metrics
- Optimization techniques

Part 2

- Supervised Learning Methods
  - Logistic Regression
  - Naive Bayes
  - k-Nearest Neighbours
  - SVM

Part 3

- Anomaly Detection

Data, data, data……

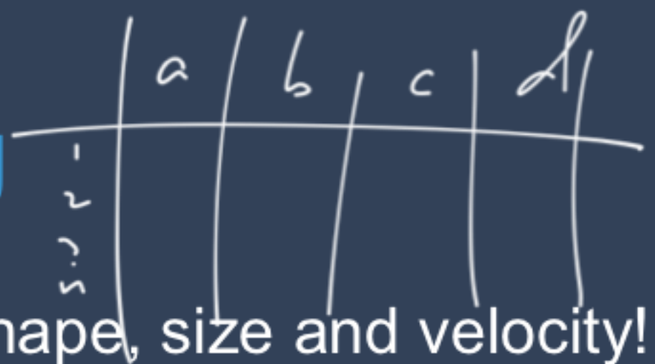
According to the World Economic Forum, an astonishing 90% of the world's data has been generated in the last two years alone. It says that 2.5 quintillion (10^{18}) bytes of data are produced by humans every day in 2020 and 463 exabytes (i.e. quintillion bytes) of data will be generated each day by humans by 2025!

But is all of this data really useful for ML algorithms and AI systems?
How do we make data more meaningful?

“Garbage in, Garbage out”

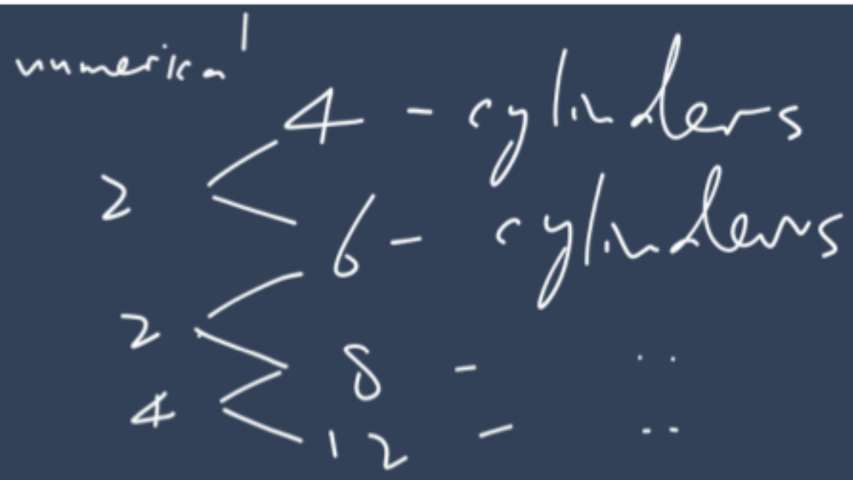
Before all the ML work, make sure your data is just relevant and logically correct.

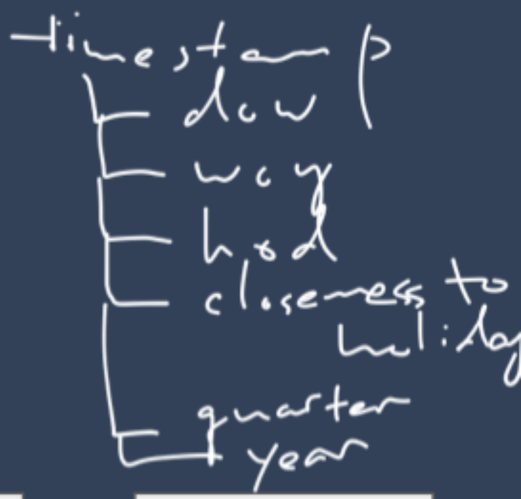
Data Preprocessing



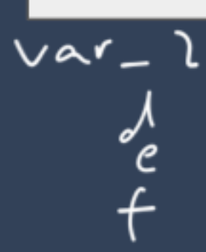
- Data could take various shape, size and velocity!
  - Shape: Structured, Unstructured (Image, Audio, Video, Table, system logging, etc.)
  - Size: a dozen rows, 1M rows, ,....
  - Velocity: Streaming data, Transaction data, Daily summary, etc.
- But machines don't understand this data out-of-the-box due to difference in representations, encoding and data type
- *Data Preprocessing* transforms/encodes data to bring it to such a state that now the machine can easily parse it
  - In other words, the features of the data can now be easily interpreted by the ML algorithm

Data Preprocessing





Raw Data	Data Type Identification	Data Quality Correction	Feature Engineering	Dimensionality Reduction	Feature Encoding
1. Structured 2. Unstructured (xx2vec)	1. *Categorical* Nominal Ordinal 2. *Numerical* Interval Ratio	1. Missing 2. Inconsistent 3. Duplicate 4. Outlier removal	1. Combination 2. Transformation	1. Sampling 2. Feature selection	1. One-hot encoding 2. Label encoding 3. Interval encoding



Data Preprocessing

Feature1	Feature2	Feature3
Male	0.53	1
Female	0.25	NULL
Male	1.5	3
Female	999.9	5

Categorical

Numerical

Outlier

Nominal/Ordinal

Missing

- 1-hot encoding
  - Male - [0, 1]
  - Female - [1, 0]
- Label encoding
  - Male - 1
  - Female - 0

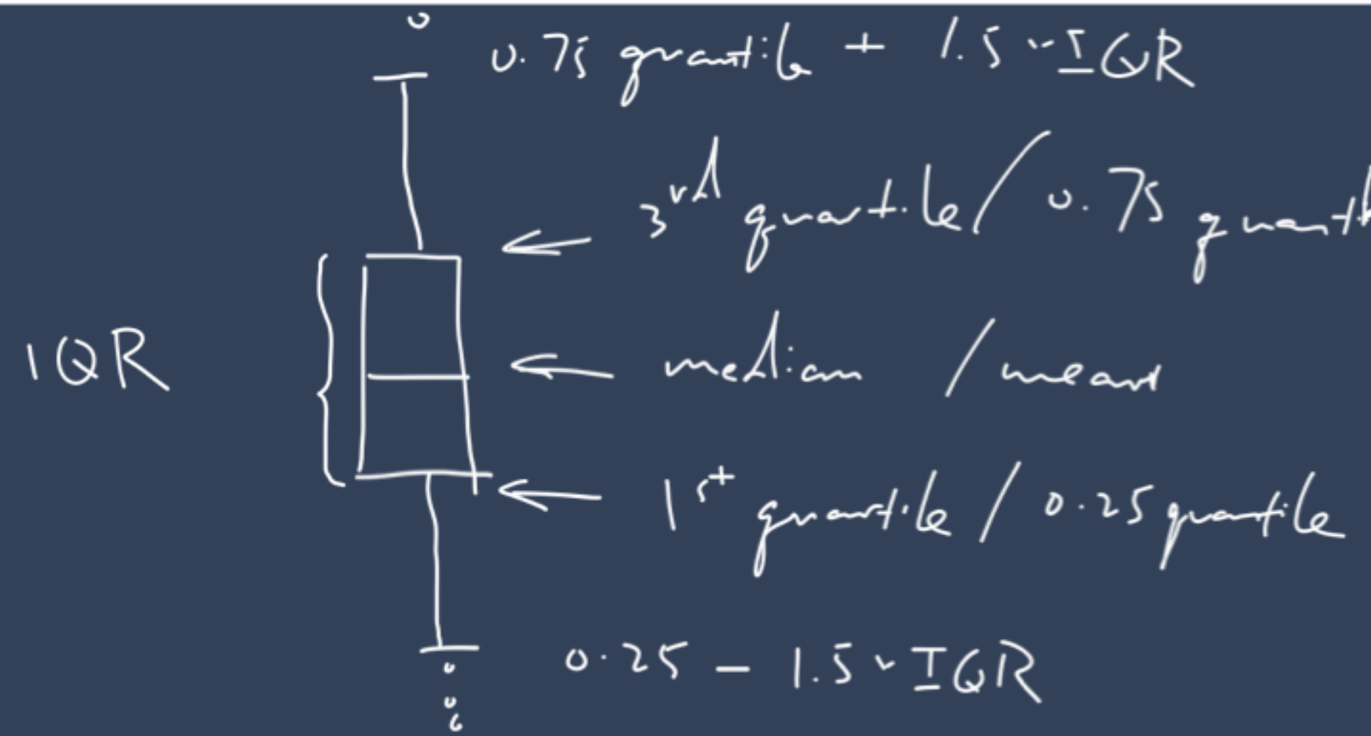
Question: Should we always remove outliers from our dataset during the preprocessing step?

a. Yes
b. No

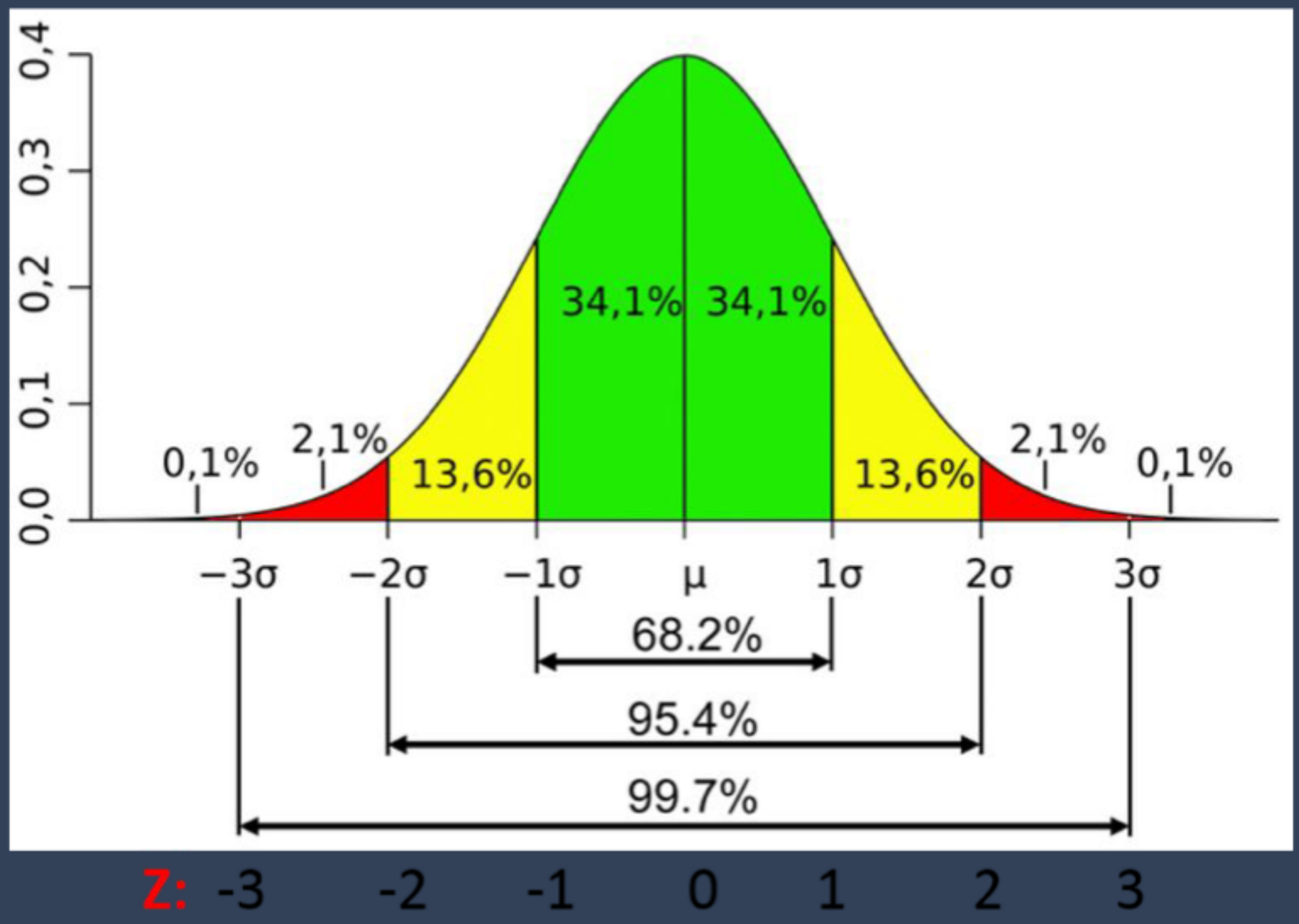
depends

Data Quality Correction

1. Imputation
   a. Median imputation
   b. Mode imputation
   c. Gaussian imputation
2. Feature dropping
   a. Drop the feature if the number of missing values are way too much
3. Outlier removal
   a. There are many problems when outliers are not desired and hence removed
   b. Most common way is to remove values beyond +/-2* IQR or beyond 99 percentile



Imputation/Outlier Removal - Example



{ik} INTERVIEW KICKSTART

Now when we have made our data meaningful, what's next? How can ML algorithms consume it?

Partition the data - Data classes

- Training Data
  - Goal is to find a model that optimizes the training objective metrics
  - This is the data on which machine learning algorithms are actually trained to build a model
  - Typically this is 70-80% of the overall data
- Validation Data
  - This is the data on which hyperparameters of the model are tuned
  - Typically this is 5-10% of the overall data
- Test Data
  - This is the data on which model is evaluated before deployment and should not be used for parameter tuning
  - Typically this is 10-20% of the overall data

Question: Why do we keep aside the validation dataset?

a. Parameter estimation
b. Hyperparameter tuning ✓
c. Both a & b
d. None

Any good ML system should be able to quantify its goodness!

Regression: Objective Functions

- Regression Loss
  - Mean Square Error/Quadratic Loss/L2 Loss
    - MSE = $\sum^{n}_{i=1}(y_i - \tilde{y}_i)^2 / n$
  - Mean Absolute Error/L1 Loss
    - MAE = $\sum^{n}_{i=1}|y_i - \tilde{y}_i| / n$

$R^2 = \frac{\text{var explain}}{\text{var in } y} = \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$

0.5 - quantile

y_i is the true target, while $\tilde{y}_i$ is the model generated target, n is the number of *training* sample

if we keep outliers . and train with it

↓

Question: Which is more robust to data with outliers?

a. MSE
b. MAE ✓
c. RMSE
d. None

Regression: Evaluation Metrics

- All the metrics in previous slides
  - MSE
  - MAE
- MAPE (Mean Absolute Percentage Error)
  - MAPE = $1/n \sum | (y_i - \tilde{y}_i) / y_i |$
  - Mean Absolute Percentage Error is also commonly used in evaluation time-series forecasting models

y_i is the true target, while $\tilde{y}_i$ is the model generated target, n is the number of *validation/testing* sample

Classification: Objective Functions

- Classification Loss
  - Cross Entropy Loss
    - Cross Entropy Loss = $-\sum(y_i \log(\tilde{y}_i) + (1 - y_i) \log(1 - \tilde{y}_i))$
    - Used in binary and multi-class classification algorithms such as logistic regression, neural networks

Classification: Evaluation Metrics

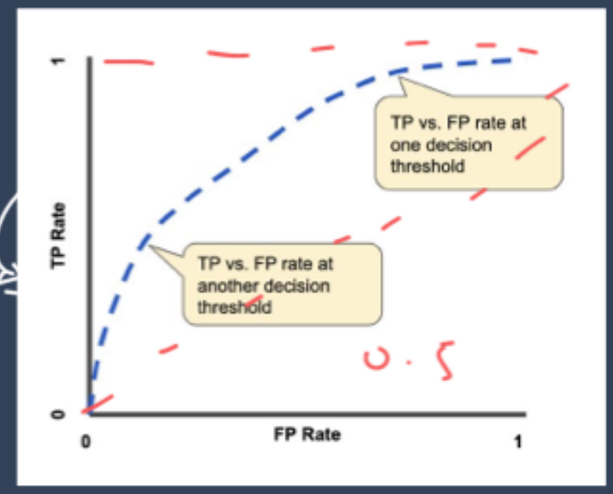
	Actual Values		
Predicted Values		Positive(1)	Negative(0)
	Positive(1)	TP	FP
	Negative(0)	FN	TN

- Confusion Matrix
  - It is a performance measurement for machine learning classification problem where output can be two or more classes.
  - Confusion table for 2 classes classification problem has 4 different combinations of predicted and actual values
  - It is extremely useful for measuring Recall, Precision, Specificity, Accuracy, and most importantly AUC-ROC curves

Classification: Evaluation Metrics

- Precision
  - TP / (TP + FP)
  - Represents % of predictions that are actually positive out of all we have predicted as positive
- Recall
  - TP / (TP + FN)
  - Represents coverage of the model in terms of how many we predicted correctly from all the positive classes
- F1-Score
  - 2 * Recall * Precision / (Recall + Precision)
- Accuracy
  - (TP + TN) / (TP + FP + TN + FN)
  - Percentage of correct predictions

Classification: Evaluation Metrics



- AUC-ROC
  - Area under curve - receiver operating characteristic curve
  - An ROC curve is a graph showing the performance of a classification model at all classification **thresholds**.
  - This curve plots two parameters:
    - Y axis -> True Positive Rate = TP / (TP + FN)
    - X axis -> False Positive Rate = FP / (FP + TN)
  - Ranges between 0 and 1 with higher the better model performance
  - For a model doing random prediction, AUC-ROC is 0.5
  - For imbalance classes, AUC-ROC may not be the best metric. This is because a small number of correct/incorrect prediction can result in a large change in the ROC curve

Confusion Matrix

	Actual Values		
Predicted Values		Positive(1)	Negative(0)
	Positive(1)	100	25
	Negative(0)	10	100

- Precision = 100 / (100 + 25) = 100 / 125 = 0.80
- Recall = 100 / (100 + 10) = 100 / 110 = 0.91
- F-Score = (2 * 100 / 125 * 100 / 110) / (100 / 125 + 100 / 110) = 0.85
- Accuracy = (100 + 100) / (100 + 25 + 10 + 100) = 0.85

Classification: Evaluation Metrics

- AUC-PR
  - Area under curve - Precision-Recall curves
  - It is a graph showing the performance of a classification model at all classification thresholds
  - This curve plots two parameters:
    - Y axis -> Precision = TP / (TP + FP)
    - X axis -> Recall = TP / (TP + FN)
  - Range between 0 and 1 with higher the better model performance
  - It is widely used for imbalance class problems

ML Metrics

- MSE
  - MSE = $\sum^{n}_{i=1}(y_i - \tilde{y}_i)^2 / n$
  - Mean squared error is widely used as a metric in regression and time-series forecasting models
- MAPE
  - MAPE = $1/n \sum | (A_i - F_i) / A_i |$
  - Mean Absolute Percentage Error is commonly used in evaluation time-series forecasting models

True optimization is the revolutionary contribution of modern research to decision processes - George Dantzig

Optimization

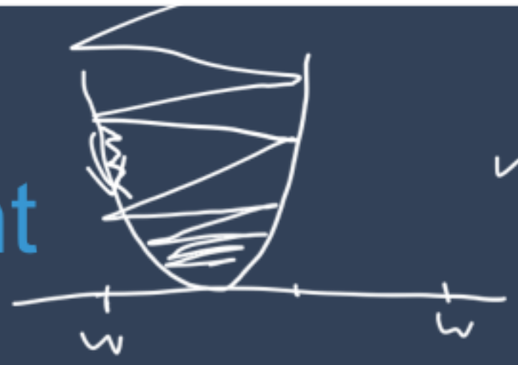
Optimization is the task of minimizing or maximizing an objective function described by a set of parameters Θ.

- For example, in linear regression, we would like to find Θ that minimize MSE i.e. $L(\Theta) = \sum^{n}_{i=1}(y_i - \text{dot}(\Theta, x_i))^2 / n$
- Linear regression with one dimension feature $L(\Theta) = \sum^{n}_{i=1}(y_i - (w_1 x_i + w_0))^2 / n$. Here $\Theta = [w_0, w_1]$

The following are the most common optimisation methods(and facilitators) used in various ML/DL algorithms:

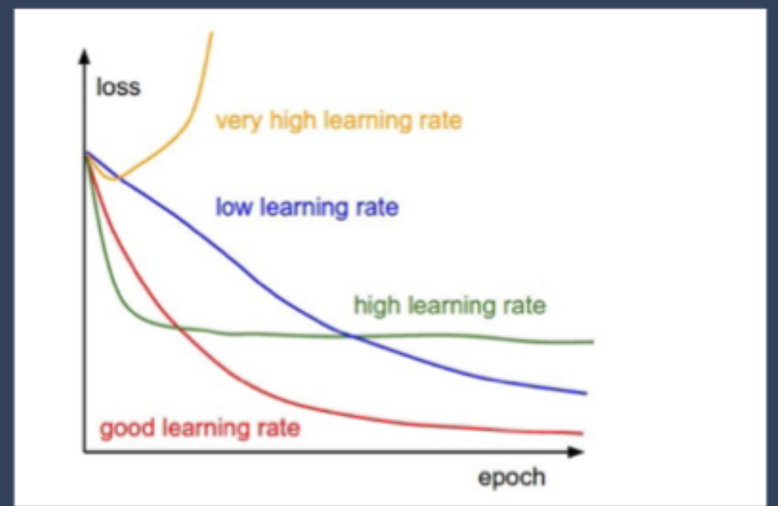
- Gradient Descent
- Stochastic Gradient Descent(SGD)
- Adam Optimiser

Gradient Descent



max increasing direction

- The goal of Gradient Descent is to minimize the objective convex function L(.) via multiple iterations(epochs). It is implemented as follows:
  a. Randomly initialize values of all the parameters
  b. At step t, update values using-
     - $w_{t+1} = w_t - learning\_rate * (\delta J(\Theta)) / (\delta weight)$
     - In our 1D linear regression example
       - $\delta L(\Theta)) / (\delta w_{1t}) = -2/n * \sum^{n}_{i=1} x_i(y_i - (w_{1t}x_i + w_{0t}))$
       - $\delta L(\Theta)) / (\delta w_{0t}) = -2/n * \sum^{n}_{i=1} (y_i - (w_{1t}x_i + w_{0t}))$
  c. Repeat until convergence condition is met



- Learning rate must be chosen wisely as:
  a. if it is too small, then the model will take some time to converge.
  b. if it is too large, model will keep on missing the convergence point by jumping abnormally

Gradient Descent with Momentum

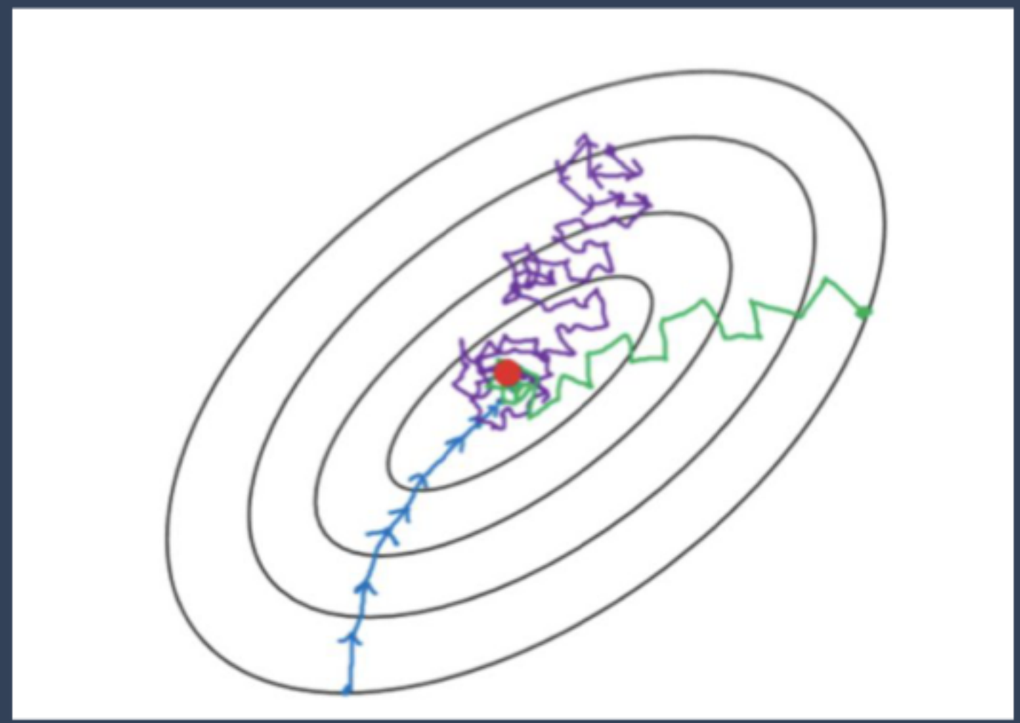
- A vanilla gradient descent model might converge slowly if the initialized values are far away from the minima
- To overcome this, we introduce the concept of momentum which accelerates the overall optimization process
- Momentum involves adding an additional hyperparameter that controls the amount of gradient history (momentum) to include in the update equation
  - $m_t = \beta m_{t-1} + (1- \beta)* (\delta L(\Theta)) / (\delta weight)$
  - $w_{t+1} = w_t - learning\_rate * m_t$
  - m_t term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

Stochastic Gradient Descent(SGD)

- The vanilla gradient descent algorithm computes gradient for each weights using all data point which makes it very slow
- Stochastic Gradient Descent helps in this case by being “stochastic”
  - In plain terms, stochastic means random
  - In our 1D linear regression example
    - $\delta L(\Theta)) / (\delta w_{1t}) = -2/n * \sum^{n}_{i=1} x_i(y_i - (w_{1t}x_i + w_{0t}))$
    - $\delta L(\Theta)) / (\delta w_{2t}) = -2/n * \sum^{n}_{i=1} (y_i - (w_{1t}x_i + w_{0t}))$
- Stochastic Gradient Descent randomly picks one data point from the whole data set at each iteration to reduce the computations enormously
- Now the above step can also be done in batches, in which case, we call it *Batch Gradient Descent*

blue: all points
smooth

green:



batch	min batch	sgd
jitter	<	jitter
smooth		

Question: Which of the following statements is NOT TRUE?

a. Purple: Stochastic Gradient Descent, Blue: Batch Gradient Descent
b. Blue: Stochastic Gradient Descent, Purple: Batch Gradient Descent
c. Blue: Batch Gradient Descent, Green: Mini Batch Gradient Descent
d. Blue: Batch Gradient Descent, Purple: Stochastic Gradient Descent

Adam Optimizer

- Adam - Adaptive Moment Estimation
- It is an algorithm for solving optimization problems on top of gradient descent algorithm
- Adam optimizer has two components to it:
  - Momentum
  - Root Mean Square Propagation (RMSP)
- Adam optimizer control the rate of gradient descent in such a way that there is minimum oscillation when it reaches the global minimum while taking big enough steps so as to pass the local minima obstacles along the way

Adam Optimizer

- Momentum
  - Finer control on *gradients*
  - $w_{t+1} = w_t - \alpha \mathbf{m_t}$
  - $m_t = \beta m_{t-1} + (1 - \beta) (\delta L / \delta w_t)$
    - m_t = aggregate of gradients at time t
    - m_{t-1} = aggregate of gradients at time t-1
    - W_t = weights at time t
    - W_{t+1} = weights at time t+1
    - α_t = learning rate at time t
    - ∂L = derivative of Loss Function
    - ∂W_t = derivative of weights at time t
    - β = Moving average parameter

Adam Optimizer

- Root Mean Square Propagation (RMSP)
  - Finer control on *learning rates*
  - $w_{t+1} = w_t - \alpha_t / (v_t + \varepsilon)^{½} * (\delta L / \delta w_t)$
  - $v_t = \beta v_{t-1} + (1 - \beta) (\delta L / \delta w_t)^2$
    - W_t = weights at time t
    - W_{t+1} = weights at time t+1
    - α_t = learning rate at time t
    - ∂L = derivative of Loss Function
    - ∂W_t = derivative of weights at time t
    - V_t = sum of square of past gradients
    - β = Moving average parameter
    - ε = A small positive constant

Adam Optimizer

- $m_t = \beta_1 m_{t-1} + (1 - \beta_1) (\delta L / \delta w_t)$
- $v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\delta L / \delta w_t)^2$
- $w_{t+1} = w_t - \alpha_t / (v_t + \varepsilon)^{½} * m_t$

Commonly used values: 0.9 for β_1, 0.999 for β_2, 10^-8 for ϵ, and 10^-3 for α_t

Time for some action now!

{ik} | INTERVIEW KICKSTART

$y = \{0, 1\}$

$y \in \mathbb{R}$

Question: Can we use Logistic Regression to solve regression problems?

a. Yes

b. No

$$y = \frac{1}{1 + \exp(-X\beta)}$$

$$\frac{1}{y} = 1 + \exp(-X\beta)$$

$$z = X\beta$$

$$X\beta = -\log\left(\frac{1}{y} - 1\right) = z$$

Question: Is logistic regression a linear model given that the decision boundary is non- linear?

X^{β}

a. Yes ✓

b. No

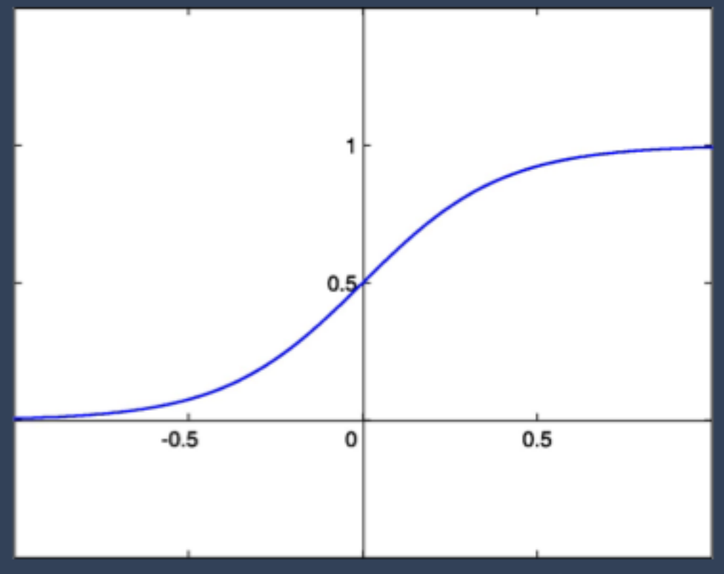
Logistic Regression

$\mathbb{R} \rightarrow (0, 1)$

- Logistic Regression (LR) is primarily used as binary classification algorithm which classifies observations into binary class 0/1
- The core component of a LR algorithm is the “sigmoid” function which acts as a probabilistic component on top of the linear equation
- Consider a single input observation x, which we will represent by a vector of features $[x_1, x_2, ..., x_n]$
- The classifier output *y* as a probability, can be thresholded to either 1 or 0 represented by either P(Y=1 | x) or P(Y=0 | x)
- Logistic Regression helps in accomplishing the above objective

Logistic Regression

- Logistic regression solves the binary classification task by learning, from a training set, a vector of **weights** and a **bias** term
- Before jumping on the classification, let's recall our regression equation
  - $z = (\sum^{n}_{i=1} w_i * x_i) + b$, or
  - $z = w \,.\, x + b$, represented by the dot product
- Now, recall the sigmoid function $\frac{e^z}{e^z + 1}$
  - $\sigma(z) = 1 / (1 + e^{-z})$
  - Sigmoid function squeezes the value between 0 and 1 and is ideal for representing the values as probability



Logistic Regression

- Hence, we easily move from regression -> classification
  - $z = w \cdot x + b$
  - $y_{hat} = \sigma(z) = 1 / (1 + e^{-z})$
  - $y_{hat} = 1 / (1 + e^{-(w.x + b)})$
- Thus,
  - $y_{hat} = 1$ if $P(y=1 \mid x) > 0.5$ (or some other thresholds)
    $= 0$ otherwise

$$p_i = y = \frac{1}{1 + e^{-X_i\beta}}$$

$p_i(1-p_i)$

each observation $i \sim Ber(p_i)$

$$X_i\beta = \log \frac{p_i}{1-p_i}$$

$z_i \sim var(p_i)$

$$\frac{1}{p_i} = 1 + e^{-X_i\beta}$$

$$X_i\beta = -\log \frac{1-p_i}{p_i}$$

$$= \log \frac{p_i}{1-p_i}$$

Logistic Regression

- How do we learn then?

- Well, Logistic Regression uses *cross-entropy* loss function to optimize for the parameter w

- We need a loss function that expresses, for an observation x, how close the classifier output is to the correct output (y, which is 0 or 1)
  - $L(y_{hat}, y)$ = How much y_{hat} differs from the true y

Logistic Regression

- We'd like to learn weights that maximize the probability of the correct label p(y|x). The probability of this can be expressed like a Bernoulli equation
  - $p(y \mid x) = y_{hat}^{y}(1 - y_{hat})^{1-y}$
- Now, we take log on both sides
  - $\log p(y \mid x) = \log (y_{hat}^{y}(1 - y_{hat})^{1-y})$
  - $\log p(y \mid x) = y \log y_{hat} + (1 - y) \log (1 - y_{hat})$
- In order to turn this into loss function, we flip the sign
  - $L = - \log p(y \mid x) = - (y \log y_{hat} + (1 - y) \log (1 - y_{hat}))$

log loss = negative log likelihood

- The above loss function can be optimized using gradient descent by taking the derivatives

Logistic Regression

- Logistic Regression could also suffer from overfitting and hence we need to regularize
- To avoid overfitting, a new regularization term R(θ) is added to the objective function, $\theta = [w_i, b]$
  - $\theta_{hat} = \text{argmax}_{\theta} \sum \log P(y^{(i)} \mid x^{(i)}) - \alpha R(\theta)$
- Here, R could be a L1 or L2 regularization function where former does feature selection while latter prevents large weights

Logistic Regression - Step by step

X1	X2	Y
3	3	0
1	2	0
3	4	0
1	2	0
3	3	0
8	3	1
5	2	1
7	2	1
9	0	1
8	4	1

- Calculate a prediction using the current values of the coefficients

Let's start off by assigning 0.0 to each coefficient and calculating the probability of the first training instance that belongs to class 0.

$b_0 = 0.0$
$b_1 = 0.0$
$b_2 = 0.0$

The first training instance is: $x_1 = 3$, $x_2 = 3$, $Y = 0$

Using the above equation we can plug in all of these numbers and calculate a prediction:

prediction = 1 / (1 + e^(-(b_0 + b_1*x_1 + b_2*x_2)))
prediction = 1 / (1 + e^(-(0.0 + 0.0*3 + 0.0*3)))
prediction = 0.5

Logistic Regression - Step by step

X1	X2	Y
3	3	0
1	2	0
3	4	0
1	2	0
3	3	0
8	3	1
5	2	1
7	2	1
9	0	1
8	4	1

- Calculate new coefficient values based on the error in the prediction

We can calculate the new coefficient values using a simple update equation.

b = b + alpha * (y – prediction) * prediction * (1 – prediction) * x

alpha is the learning rate and controls how much the coefficients (and therefore the model) changes or learns each time it is updated. We would use a value of 0.3 for this example.

Let's update the coefficients using the prediction (0.5) and coefficient values (0.0) from the previous section.

$b_0 = b_0 + 0.3 * (0 – 0.5) * 0.5 * (1 – 0.5) * 1$

$b_1 = b_1 + 0.3 * (0 – 0.5) * 0.5 * (1 – 0.5) * 3$

$b_2 = b_2 + 0.3 * (0 – 0.5) * 0.5 * (1 – 0.5) * 3$

or

$b_0 = -0.0375$, $b_1 = -0.1125$, $b_2 = -0.1125$

Logistic Regression - Example

Example: Suppose that we are interested in the factors that influence whether a politician wins an election. The outcome variable is binary (0/1); win or lose.

The predictor variables of interest are:

- amount of money spent on the campaign
- amount of time spent campaigning negatively
- whether or not the candidate is an incumbent

Naive Bayes

- Naive Bayes is a supervised probabilistic algorithm that utilizes probability theory and Bayes' Theorem to predict probability of occurrence of an event
- The algorithm assumes that each input variables is **independent** which is a *naive assumption* in reality and hence called as Naive Bayes
- For example,
  - if you use Naive Bayes for sentiment analysis, given the sentence 'I like Game of Thrones', the algorithm will look at the individual words and not the full sentence
  - In a sentence, words that stand next to each other influence the meaning of each other, and the position of words in a sentence is also important
  - However, for the algorithm, phrases like 'I like Game of Thrones', 'Game of Thrones like I', and 'Thrones I like of Game' are the same

Naive Bayes

- To make classifications, we need to use X to predict Y. In other words, given a data point $X = (x_1, x_2, \ldots, x_n)$, what the odd of this sample belongs to y. This can be rewritten as the following equation:
  - $P(Y = y \mid X = (x_1, x_2, \ldots, x_k))$
- Bayes Theorem
  - $P(Y \mid X) = (P(X \mid Y) * P(Y)) / P(X)$ → Posterior = (likelihood * prior) / evidence
  - We can't get P(Y | X) directly but we can get P(X | Y) and P(Y) from the data

$$P(X \mid Y) = P(X_1, X_2, \cdots X_n \mid Y)$$
$$= P(X_1 \mid Y)\, P(X_2 \mid Y) \cdots P(X_n \mid Y)$$

Naive Bayes

- As the number of feature grows, it becomes complex to calculate P(X | Y). For example with 2 features(X_1 & X_2), we would need to calculate following,
  - $P(X_1 = 1, X_2 = 1 \mid Y = 1)$, $P(X_1 = 1, X_2 = 1 \mid Y = 0)$
  - $P(X_1 = 1, X_2 = 0 \mid Y = 1)$, $P(X_1 = 1, X_2 = 0 \mid Y = 0)$
  - $P(X_1 = 0, X_2 = 1 \mid Y = 1)$, $P(X_1 = 0, X_2 = 1 \mid Y = 0)$
  - $P(X_1 = 0, X_2 = 0 \mid Y = 1)$,
  - $P(X_1 = 0, X_2 = 0 \mid Y = 0) \rightarrow$ 1 - others
- There are $2^{k+1} - 1$ parameters where *k* is the number of features
- Solving for these number of parameters is time-consuming and hence we introduce the independence assumption!
  - If X_1 and X_2 are independent, $P(X_1, X_2 \mid Y) = P(X_1 \mid Y) * P(X_2 \mid Y)$

Naive Bayes

- Naive Bayes Classifier:
  - $P(Y \mid X) = P(Y) \prod_i P(X_i \mid Y) / P(X)$
- For continuous features, there are two possible routes:
  - Discretization - Self explanatory
  - Continuous Naive Bayes
- In continuous Naive Bayes, we first assume the probability distribution from the following
  - Gaussian
  - Multinomial
  - Bernoulli
- And then, estimate the parameters of these distributions using the data to get the PDF

Question: Naive Bayes is not suitable for datasets with large numbers of numerical attributes.

a. True

b. False

Naive Bayes Example

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	NO
sunny	hot	high	true	NO
overcast	hot	high	false	YES
rainy	mild	high	false	YES
rainy	cool	normal	false	YES
rainy	cool	normal	true	NO
overcast	cool	normal	true	YES
sunny	mild	high	false	NO
sunny	cool	normal	false	YES
rainy	mild	normal	false	YES
sunny	mild	normal	true	YES
overcast	mild	high	true	YES
overcast	hot	normal	false	YES
rainy	mild	high	true	NO

Weather dataset, from the University of Edinburgh

- In this case, X = (Outlook, Temperature, Humidity, Windy), and Y = Play.
  - Q: today = (sunny, hot, normal, false)
  - A: P(play| today) and P (No play | today)?
- P (play | today)

  = P(today | play) * P (play)/ P(today)

  = P(sunny | play) * P(hot | play) * P(normal | play) * P (false | play) * P(play) / P(today)

  = 2/9 * 2/9 * 6/9 * 6/9 * 9/14 / P(today)

  = 0.0141/P(today)
- P (Not play | today)

  = 3/5 * 2/5 * 1/5 * 2/5 * 5/14 / P(today)

  = 0.0068 / P(today)

Naive Bayes Example

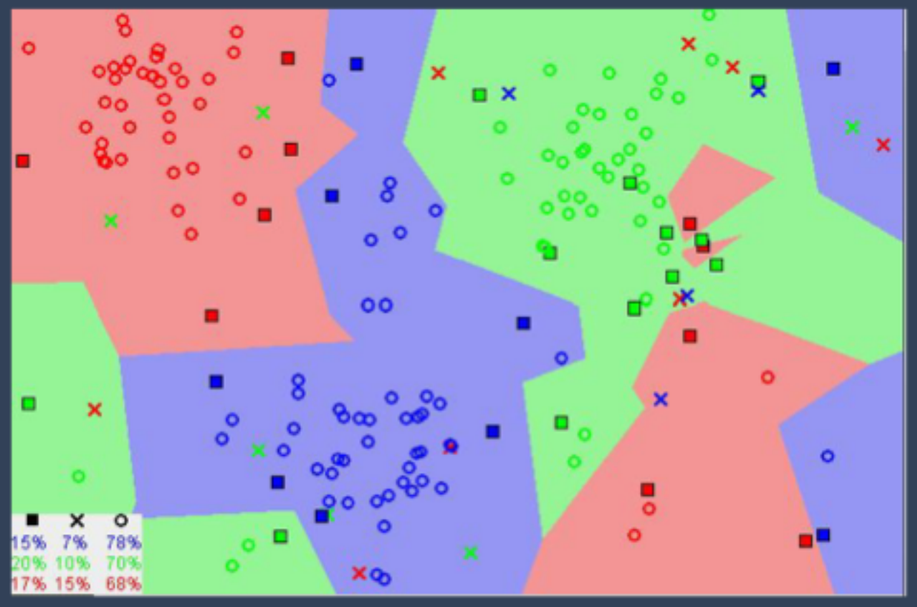
- P (play | today)
  = 0.0141/ P(today)
- P (Not play | today)
  = 0.0068 / P(today)

Conclusion:

- P (play | today) > P (Not play | today)
- P (play | today) = 0.67, P (Not play | today) = 0.33 using the fact P (play | today) + P (Not play | today) = 1

k - Nearest Neighbors (kNN)

- kNN is a supervised machine learning algorithm used in both classification and regression problems
- The basic assumption here is that similar items appear in close ***proximity*** in a n-dimensional space



k - Nearest Neighbors (kNN)

- One of the most important aspects while using a k-nn algorithm is to define the distance metric for calculating the neighbors such as
  - Euclidean
  - Cosine
- The choice of distance metric is a hyperparameter and depends on the type of dataset
- The number of neighbors *k* is also an hyperparameter which needs to be identified using the validation dataset

Question: What are the hyperparameters in kNN based modeling?

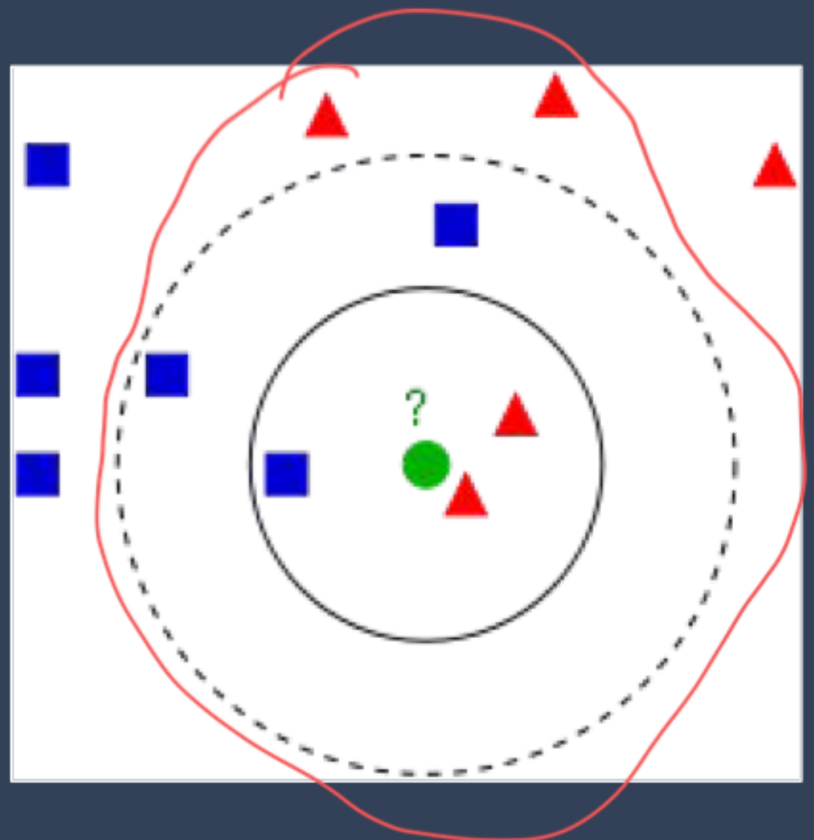
a. Distance function
b. k -> number of neighbors
c. Both a & b ✓

k - Nearest Neighbors (kNN)

Algorithm

1. Set the value of k and choose the distance measure
2. For each data point
    a. Calculate the distance between the input data point and the current data point from the data
    b. Add the distance and the index of the example to an ordered collection
3. Sort the ordered collection of distances and indices in ascending order by the distances
4. Pick the first K entries from the sorted collection and the corresponding labels
5. In case of classification problem, could return the mode of the K labels
6. In case of regression problem, could return the mean of the K values

k - Nearest Neighbors (kNN)



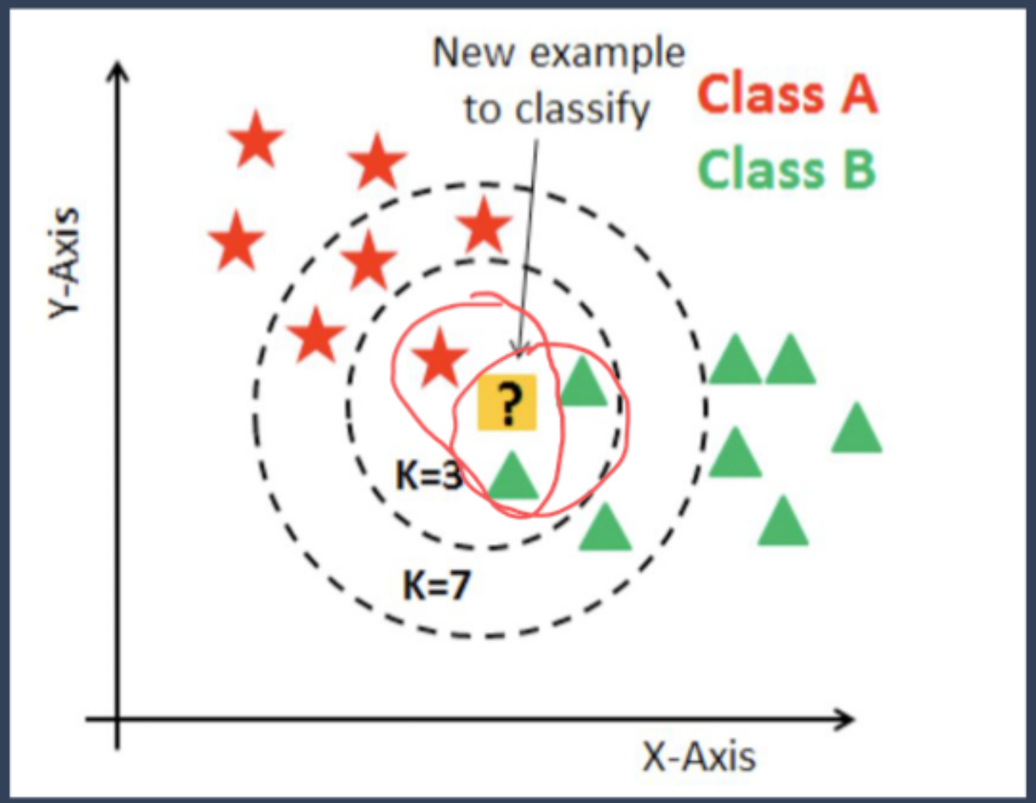
Which group (red, blue) the green dot belongs to?

k = 3 . red

k = 5 . blue

k = 7

k = odd

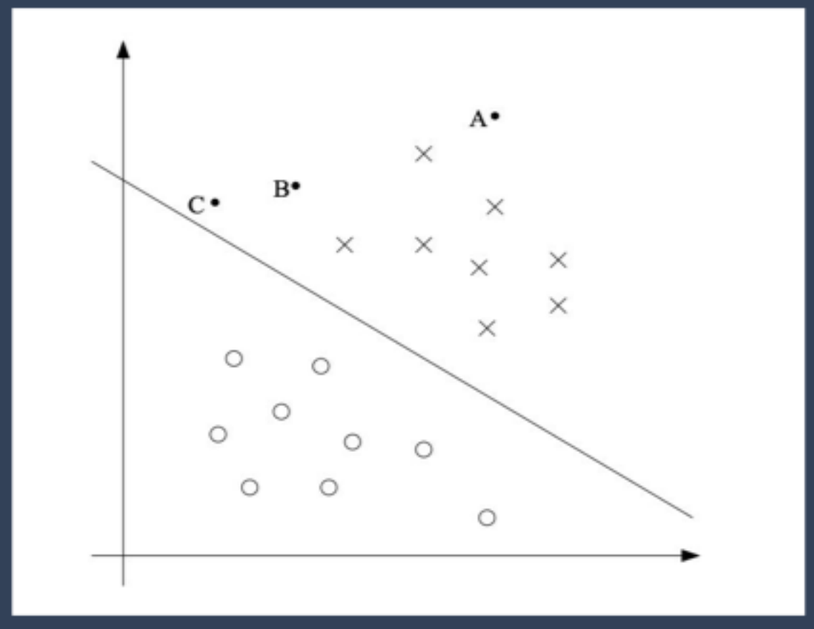


Question: In the above figure, which Class would Yellow(?) belong to when k=2?

a. Class A
b. Class B

SVM - Support Vector Machine

- SVM is a binary classification algorithm which falls in the category of maximum-margin classifier
- It is a discriminative model formally defined by a separating hyperplane. In 2-dimensional space, it is just called line.

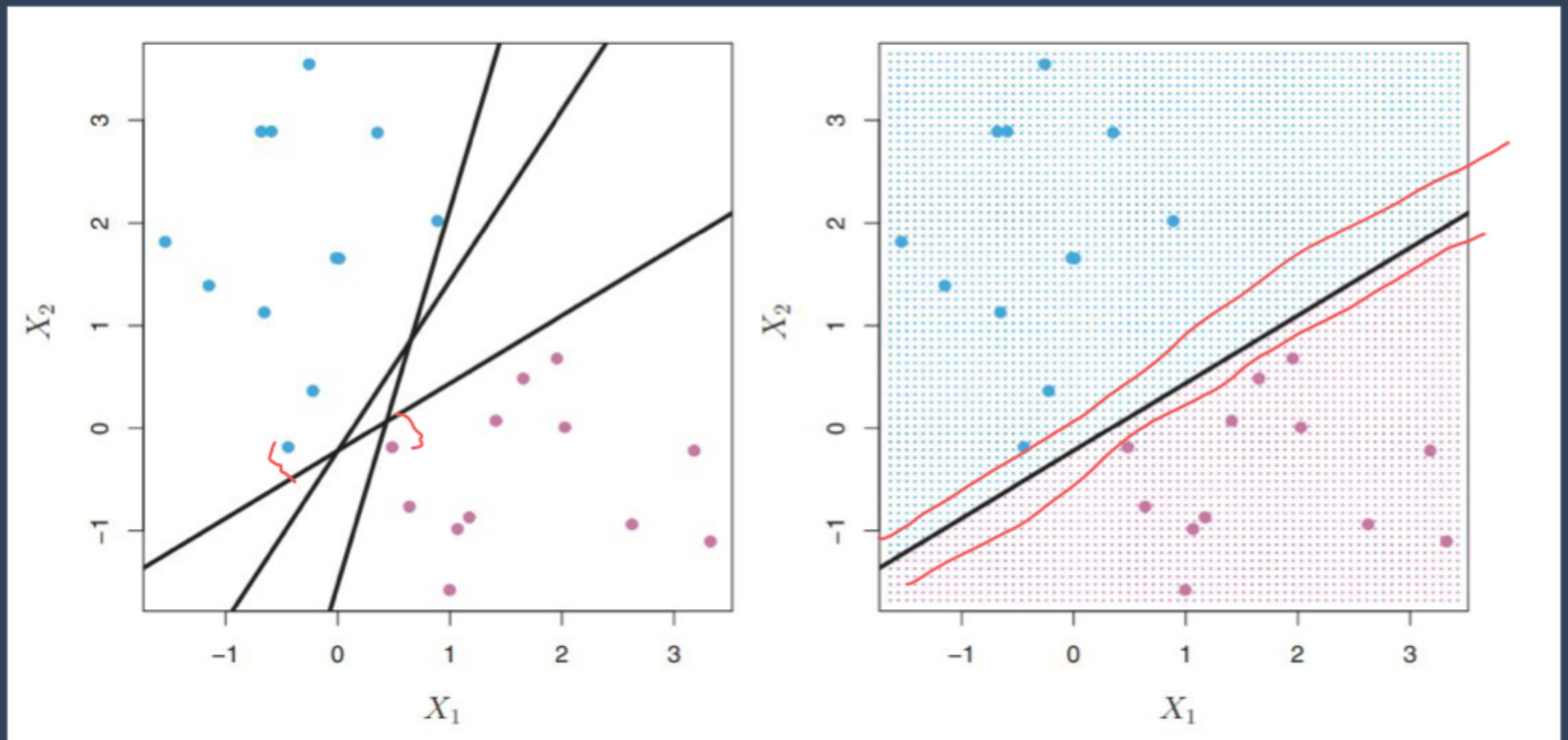


SVM - Support Vector Machine

- The decision boundary can be represented using $w^T . x + b$
- Intuitively, the further a point is from the hyperplane on either side, the more confident we are about a prediction.
- The hyperplane $w^T . x + b$ then separates classes
- Hence, we try to create the hyperplane is such a manner that margins of the points from the separating hyperplane are as high as possible

SVM maximizes the distances to the closest points on both sides

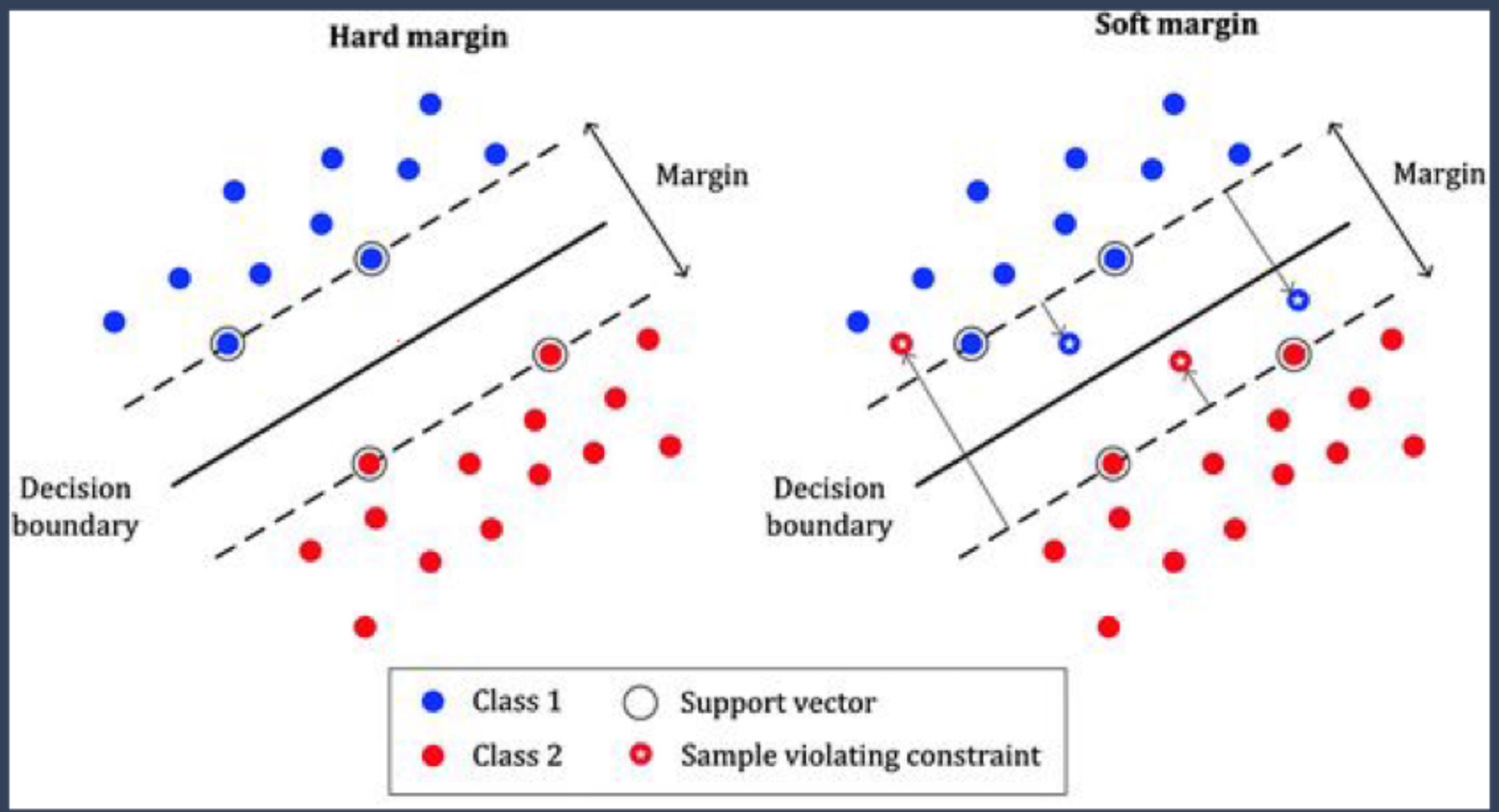
SVM - Support Vector Machine



SVM - Support Vector Machine

- Notation
  - Class label y ∈ {-1, 1}
  - $h_{w,b}(x) = g(w^T . x + b)$, *note here we separate out* θ as [w, b]
  - *Here, g(z) = 1* if $z \geq 0$, and $g(z) = -1$, otherwise
- Functional margin
  - $\gamma^{(i)} = y^{(i)} (w^T . x_i + b)$
  - if $y^{(i)} = 1$, then for the functional margin to be large we need $w^T . x^{(i)} + b$ to be a large positive number
  - if $y^{(i)} = -1$, then for the functional margin to be large we need $w^T . x^{(i)} + b$ to be a large negative number

SVM - Support Vector Machine

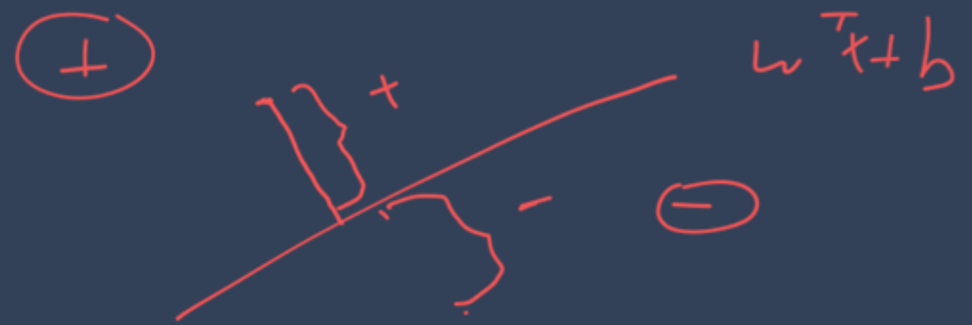


SVM - Support Vector Machine

- The magnitude of the functional margin depends on the magnitude of the parameters which is not very meaningful
  - Replacing (w, b) with (2w, 2b) results in multiplying our functional margin by a factor of 2 without changing the sign (prediction results)
  - We can add an arbitrary scaling constraints to (w, b) without changing the prediction sign
  - By exploiting our freedom to scale w and b, we can make the functional margin arbitrarily large without really changing anything meaningful

- Hence, it makes sense to impose some sort of normalization condition such as that $||w||^2 = 1$

SVM - Support Vector Machine



- Given a training set S = {($x^{(i)}$, $y^{(i)}$); i = 1, . . . , m}, we also define the function margin of (w, b) with respect to S as the smallest of the functional margins of the individual training examples.
  - $\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)}$

$\hat{\gamma}(w,b)$

(w, b)

$(2w, 2b)$

- The optimization problem of SVM can now be defined as:
  - $\max_{\gamma,w,b} \hat{\gamma}$
    - s.t. $y^{(i)} (w^T x^{(i)} + b) \geq \hat{\gamma}$, $i = 1,\ldots,m$
    - $||w|| = 1$

$\hat{\gamma}(w,b)$

$\hat{y} \in \{1, -1\}$

$\gamma(w,b)$

SVM - Support Vector Machine

- But the above formulation with ||w||=1 is non-convex and hence difficult to solve!
- What if we transform it as follows:
  - max $_{\gamma,w,b}\ \hat{\gamma}$ / ||w||
  - s.t. $y^{(i)}\ (w^T x^{(i)} + b) \geq \hat{\gamma}$, *i = 1,.....,m*
- The rational is that
  - $\hat{\gamma}^{(i)} = y^{(i)}\ (\ (w / ||w||\ )^T x^{(i)} + b / ||w||\ )$ - geometric margin
  - If ||w|| = 1, then functional margin is same as geometric margin
- The geometric margin is invariant to rescaling of the parameters

SVM -/ Support Vector Machine

- Recall: We can add an arbitrary scaling constraints to (w, b) without changing the prediction sign.
- Here, we introduce the scaling constraint that the functional margin of w, b with respect to the training set must be 1
  - $\hat{\gamma} = 1$
- Then we can transform the following as
  - ~~$\max_{\gamma,w,b} \gamma^{\Delta} / ||w||$~~
    - ~~s.t. $y^{(i)} (w^T x^{(i)} + b) \geq \gamma^{\Delta}$, $i = 1,.....,m$~~
  - $\max_{w,b} 1/ ||w||$
    - s.t. $y^{(i)} (w^T x^{(i)} + b) \geq 1$, $i = 1,.....,m$

SVM - Support Vector Machine

- Also, maximizing $1/||w||$ is same as minimizing $||w||^2$. Thus,
  - $\min_{w,b} 1/2 * ||w||^2$
    - s.t. $y^{(i)} (w^T x^{(i)} + b) \geq 1$, $i = 1,.....,m$
- The above is an optimization problem with a convex quadratic objective and only linear constraints. It's solution gives us the *optimal margin classifier*
- The Lagrangian of the above equation would help us to solve it efficiently

SVM - Support Vector Machine

Lagrangian

- Consider a problem of the following form:
  - $\min_w f(w)$
    - s.t. $h_i(w) = 0$, $i = 1, \ldots, l$
- We define the *Lagrangian* to be
  - $L(w, \beta) = f(w) + \sum_{i=1}^{l} \beta_i h_i(w)$
  - Here, the β_i's are called the *Lagrange multipliers*
- We would then find and set *L*'s partial derivatives to zero and solve for w and β

SVM - Support Vector Machine

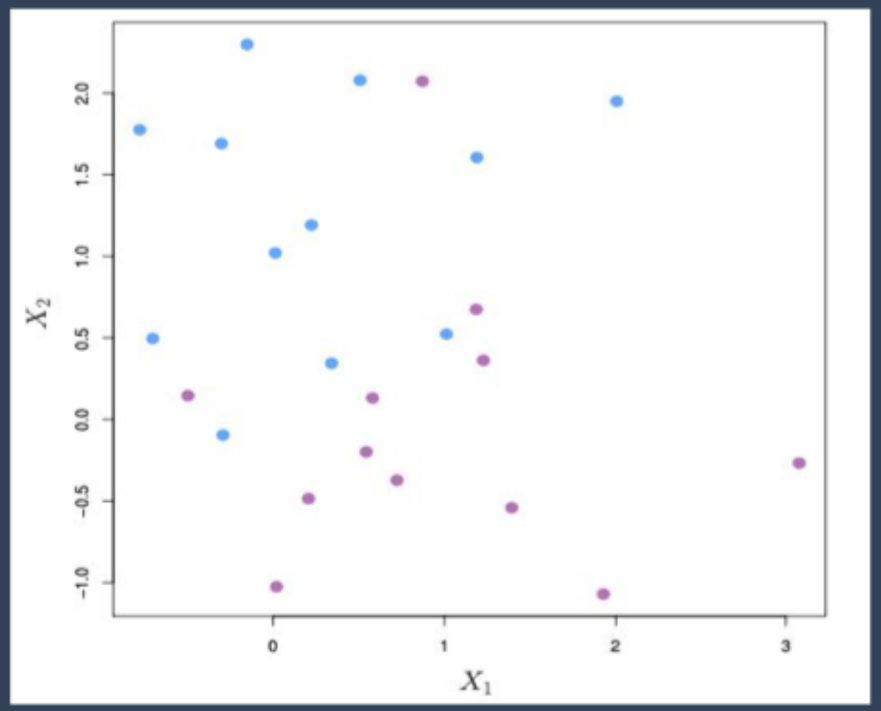
- Now remember our original optimization problem
  - $\min_{w,b} 1/2 * ||w||^2$
    - s.t., $y^{(i)} (w^T x^{(i)} + b) \geq 1$, *i = 1,.....,m*
- We can write the constraints as
  - $g_i(w) = -y^{(i)} (w^T x^{(i)} + b) + 1 \leq 0$
- When we construct the Lagrangian for our optimization problem we have:
  - $L(w, b, \alpha) = 1/2 ||w||^2 - \sum^m_{i=1} \alpha_i (y^{(i)} (w^T x^{(i)} + b) - 1)$, $\alpha_i >= 0$
  - Taking the derivative of the above equation would solve for our optimal parameters

SVM - Support Vector Machine

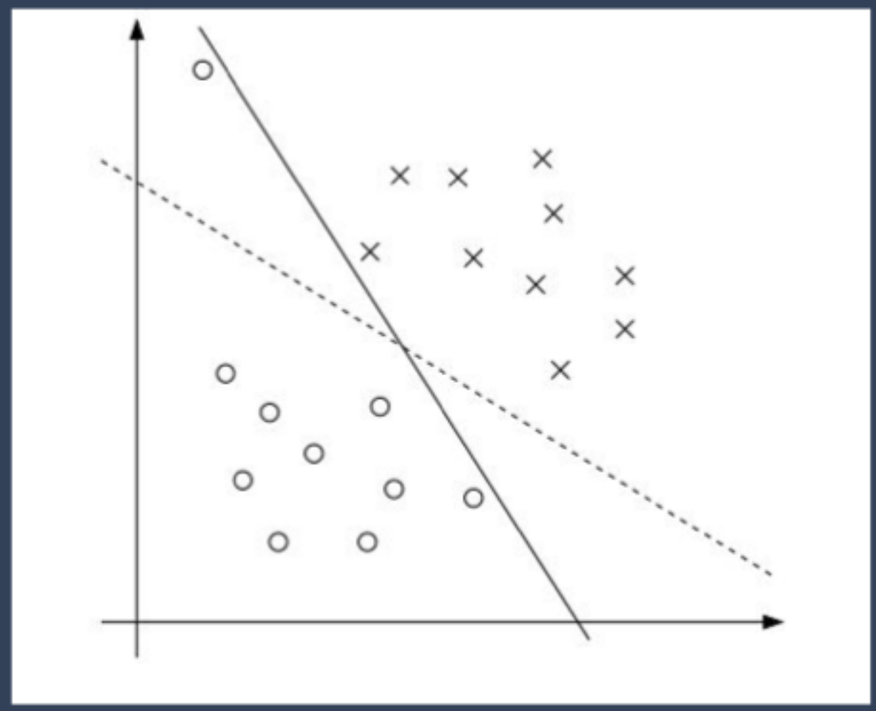
- Recall in SVM prediction, we use the sign of $\theta^T . x_{new}$ to decide the class for x_{new}
  - $\theta^T . x_{new} = w^T . x_{new} + b$
  - $w^T = \sum^m_{i=1} \alpha_i y^{(i)} x^{(i)}$
  - $w^T . x_{new} + b = \sum^m_{i=1} \alpha_i y^{(i)} x^{(i)} . x_{new} + b$

SVM - Support Vector Machine

Not separable by a hyperplane



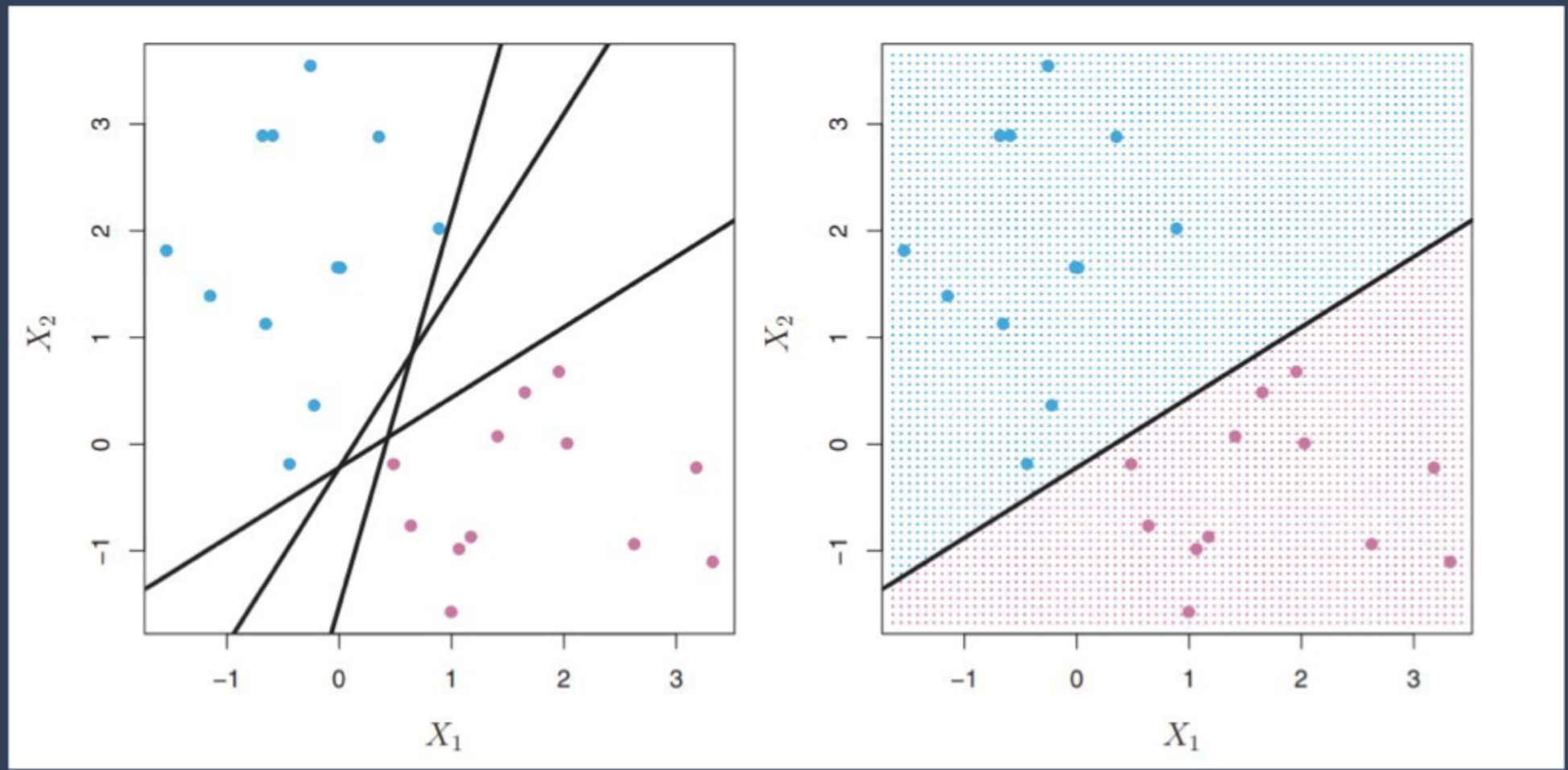
Which hyperplane is better?



SVM - Soft Margin

- The derivation of the SVM as presented so far assumed that the data is linearly separable
- To make the algorithm work for non-linearly separable datasets as well as be less sensitive to outliers, we re-formulate our optimization (using L1 regularization) as follows:
  - $\min_{\gamma,w,b} 1/2 \|w\|^2 + C\sum^{m}_{i=1}\chi_i$
    - s.t. $y^{(i)} (w^T x^{(i)} + b) \geq 1- \chi_i$, $i = 1,.....,m$
- The parameter *C* controls the relative weighting between the twin goals of making the $\|w\|^2$ large and of ensuring that most examples have functional margin at least 1

SVM - Support Vector Machine



SVM - Kernel Trick

Fig.3

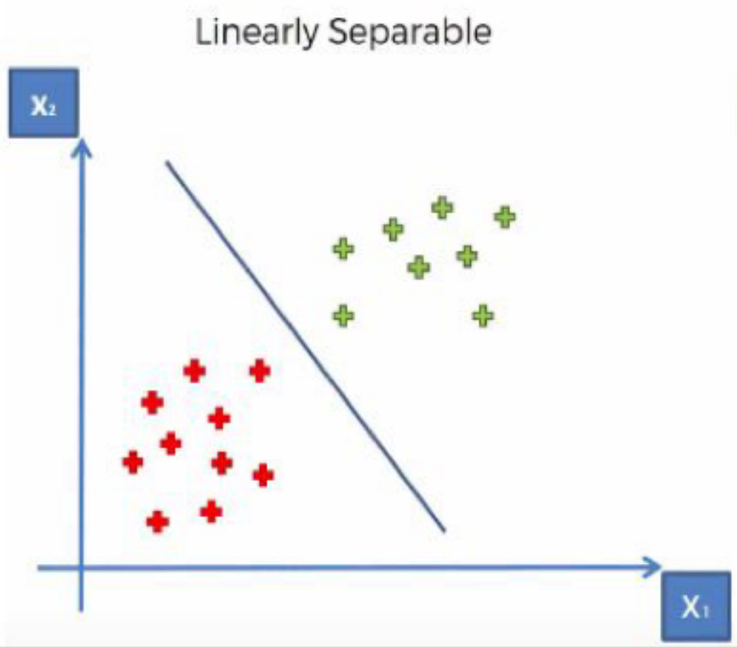
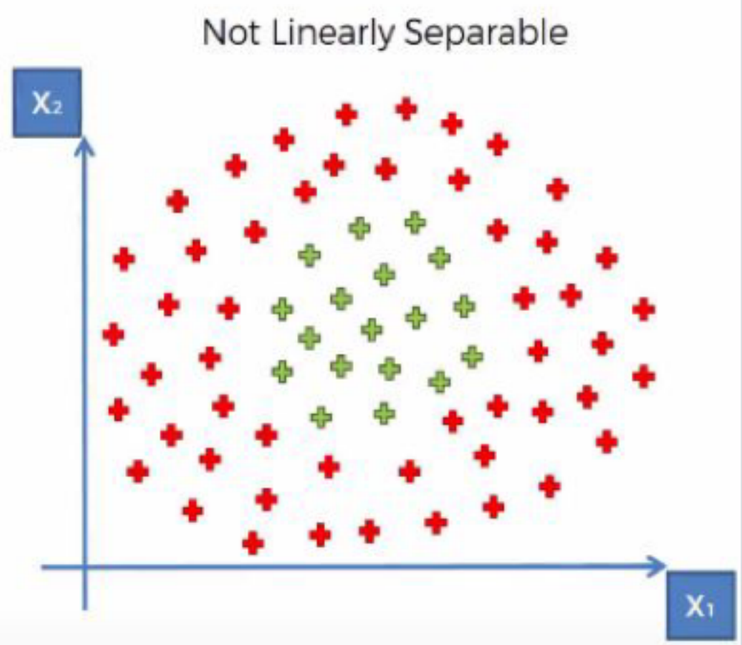
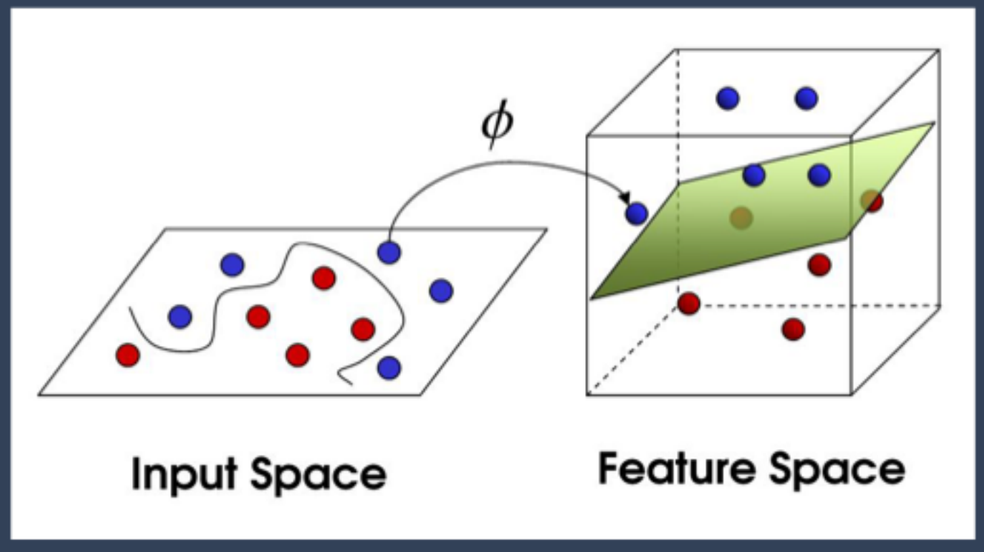


Fig.4



SVM - Kernel Trick

- The function of kernel is to take data as input and transform it into the required form, similar to *feature mapping*, we transform x to $[x \quad x^2 \quad x^3]^T$
- SVM algorithms use a set of mathematical functions that are defined as the kernel

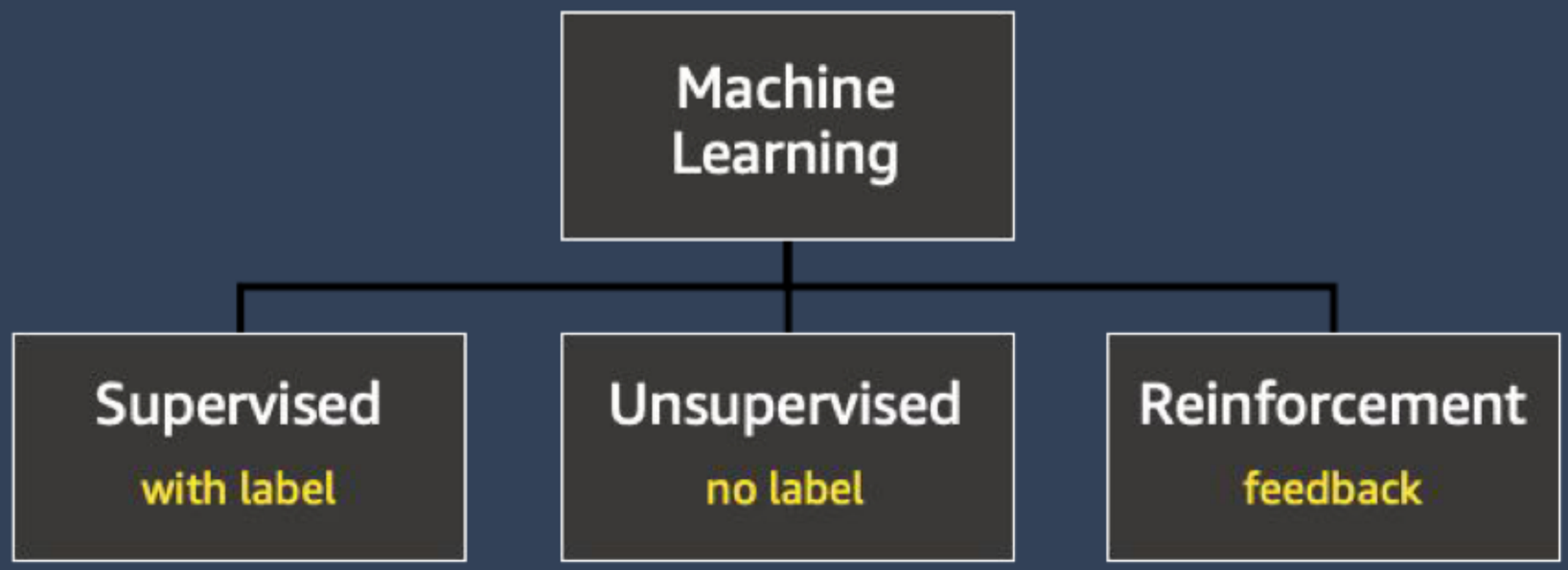


SVM - In a nutshell

1. What is hyperplane?
    a. If we have *p*-dimensional space, a hyperplane is a flat subspace with dimension *p-1* which separates point in the multi-dimensional space
2. If we divide data based on hyperplane, then there could be many possible hyperplanes to divide same set of numbers. How do we decide the finite number of hyperplanes to choose from?
    a. Maximal margin classifier
3. What if the points are non-separable?
    a. Soft-margin classifier
4. There are cases where we cannot draw a boundary that can separate two classes in linear fashion. What to do then?
    a. Kernel transformation

Anomaly Detection: A Flavor of Unsupervised Learning

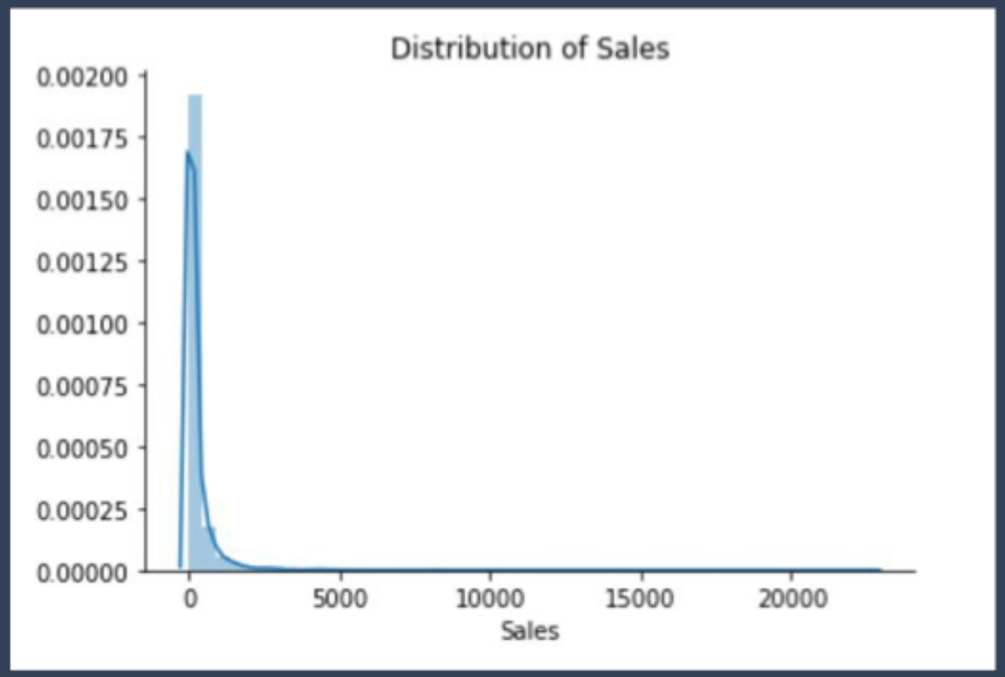
Family of Machine Learning

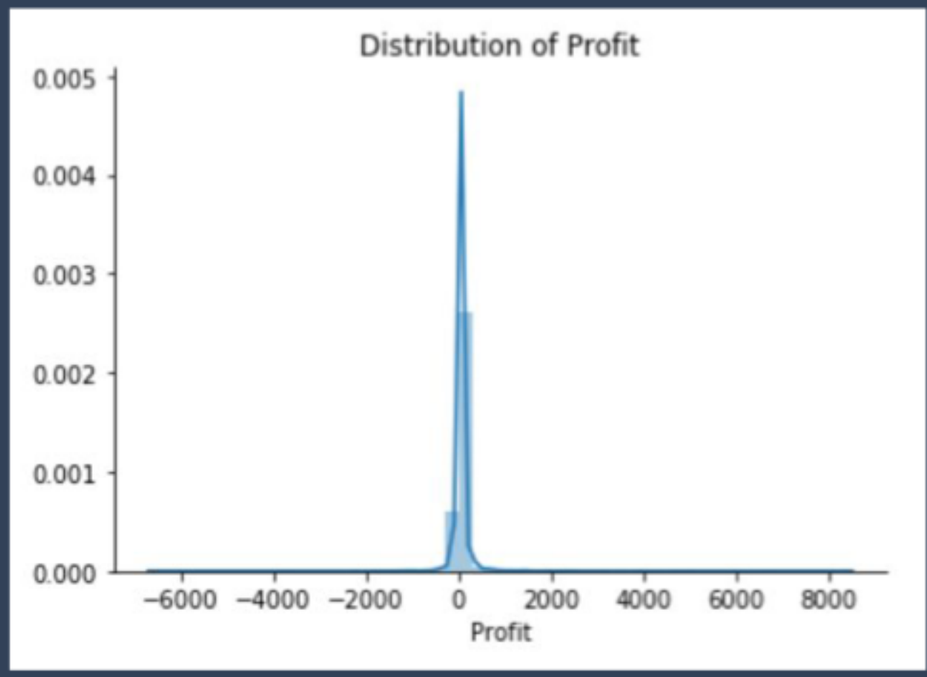


Anomaly Detection

- Anomaly detection is the process of identifying abnormal or unexpected events in datasets
- It is often applied on datasets where there are no-labels and hence it falls in the bucket of unsupervised learning
    - However, you can always train a supervised model on a set of correctly identified anomalies
- Anomaly detection can be accomplished by the following algorithms:
    - IQR
    - Isolation Forest
    - etc.

Anomaly Detection





- In the figures above, there are values on left & right extremes of the mean density which are rare in occurrence and hence often referred to as ***outliers***

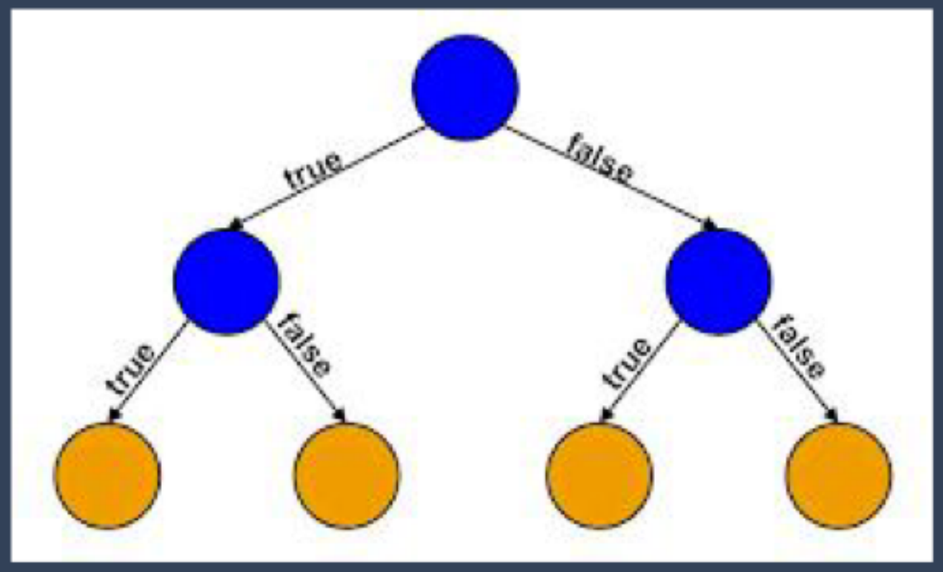
Anomaly Detection

IQR - Interquartile range

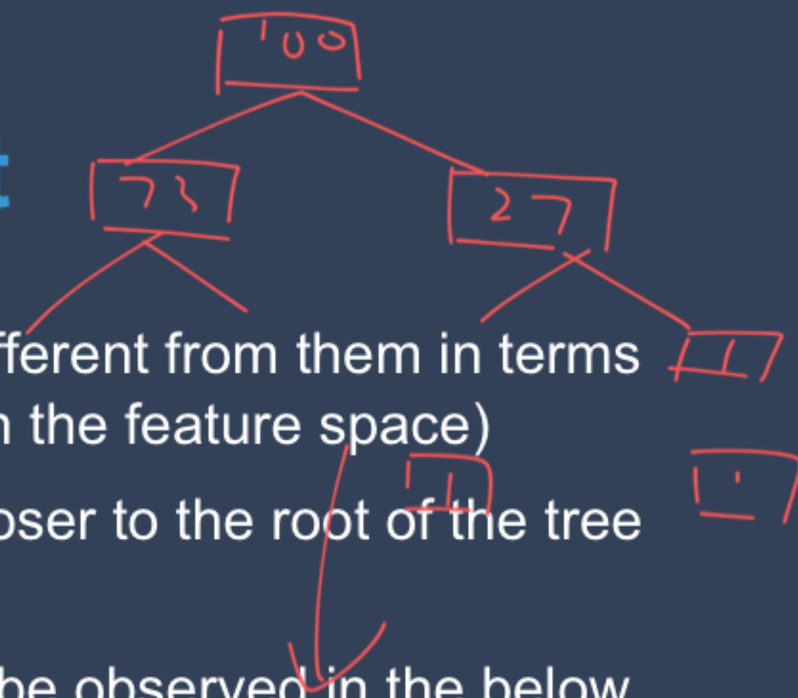
- The interquartile range defines the difference between the 3rd(Q3) and the 1st(Q1) quartile
  - IQR = Q3 - Q1
- Any value is referred to as anomaly if it satisfies any of the following 2 conditions
  - > 1.5 * IQR
  - < 1.5 * IQR
- IQR is used to identify univariate anomalies

Anomaly Detection - **Isolation Forest**

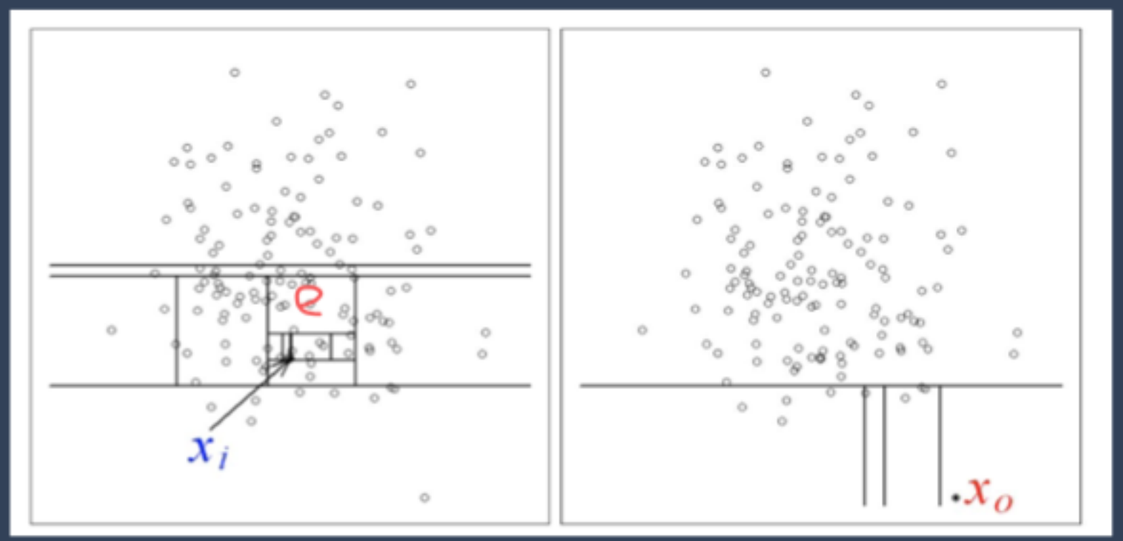
- Isolation Forest is a tree based *ensemble* algorithm to identify anomalies
- In these *trees*, partitions are created by first *randomly* selecting a feature and then selecting a *random* split value between the minimum and maximum value of the selected feature



Anomaly Detection - **Isolation Forest**



- Outliers are less frequent than regular observations and are different from them in terms of values (they lie further away from the regular observations in the feature space)
- By using such random partitioning, they should be identified closer to the root of the tree (shorter average path length), with fewer splits necessary
- The idea of identifying a normal vs. abnormal observation can be observed in the below figure [from the original paper].



A normal point (on the left) requires more partitions to be identified than an abnormal point (right).

Anomaly Detection - **Isolation Forest**

- Anomaly scored in case of Isolation Forest is defined as:
  - $s(x, n) = 2^{-E(h(x)) / c(n)}$
    - h(x) is the path length of observation x
    - c(n) is a normalized constant for a data set of size n. Typically, we use the average path length of all samples. This has a approximation as $2*(\ln(n) + 0.577) - 2 (n-1)/n$.
- Following decision can be made on the basis of anomaly score:
  - A score close to 1 indicates anomalies
    - from above formula, as *E(h(x))* goes to 0, anomaly scores goes to 1
  - Score much smaller than 0.5 indicates normal observations

Anomaly Detection

Points to note

- Anomaly detection can be done for both Univariate and Multivariate datasets
- Anomaly detection is also used in following areas:
    - Data quality monitoring
    - KPI measurement
    - Systems with long lagging feedback period
- There is an inherent assumption in many algorithms that anomalies are rare and typically present in less than 5% of the population