

Machine Learning: An Introduction



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



ML: From Rules to Data



Example: Activity Recognition



Example: Activity Recognition



0101001010100101010
1001010101001011101
0100101010010101001
0101001010100101010



1010100101001010101
0101010010010010001
0010011111010101111
1010100100111101011

Label = WALKING



1001010011111010101
1101010111010101110
1010101111010101011
1111110001111010101

Label = BIKING



111111111010011101
0011111010111110101
0101110101010101110
1010101010100111110

Label = GOLFING

Demo: WAP



- Please choose one correct answer:
- $1 \mid 3 \mid 5 \mid 7 \mid 9 \mid ?$
 - ?
 - $y = 2x - 1$
- $3 \mid 8 \mid 15 \mid 24 \mid 35 \mid ?$
 - Quiz1=
<https://forms.gle/U5MFgeYPfMKfuYBd9>
 - $y = x^*(x+2)$



Training and Testing



Training Phase

What is ML?



- Term “Machine Learning” coined by Arthur Samuel in 1959.
 - [Samuel Checkers-playing Program](#)
- Common definition (by Tom Mitchell)
 - Machine Learning is the study of computer algorithms that improve automatically through experience

More details



- Study of algorithms that
 - improve their performance P
 - at some task T
 - with experience E
- Well-defined learning task: $\langle P, T, E \rangle$



Task (T)



- Classification or Pattern Recognition
- Regression or Prediction
- Clustering
- Synthesis or Sampling
- Ranking
- Recommendation Systems
- Anomaly Detection
- Data Mining etc.



Performance (P)

- A quantitative measure to evaluate performance
 - Usually Task specific
- Classification

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

Performance (P)



- Regression
 - Error measure such as ‘mean squared error’



Experience (E)

- Supervised Learning
 - Labelled data – (Data, target value)
 - Target value could be category/class labels, real value, real vector, etc.
 - Classification, Regression
- Unsupervised Learning
 - Only data, no labels
 - Dimensionality Reduction, ICA, Clustering
- Reinforcement Learning
 - No examples, but a reward function
 - Payoff based on actions

An Incomplete History of Learning



- Turing Test (1950)
 - Machines do very poorly
- Rosenblatt's Perceptron (1960's)
 - Kick started the mathematical analysis of the learning process
 - Key idea behind Support Vector Machines (SVMs) and Neural Networks
- Construction of Fundamentals of Learning Theory (1960-70's)
 - Focus on generalization capability of learning machines
 - Performance on unseen data
 - Regularization for ill-posed problems
 - e.g., linear equations for ill-conditioned matrices
- Neural Networks (1980's)
 - Connectionism
 - Back-propagation [LeCun, '86]
 - CNNs, RNNs
- SVMs (1990's)
 - Margin Maximization
 - Kernel Methods to handle non-linearity
- Deep Learning (>2006)
 - Hinton, Bengio, LeCun at forefront
 - Abstract Representations
- (>2012) Craziness!!

Most Amazing Milestones So Far



- 1997 – Deep Blue defeats world chess champion Garry Kasparov

Deep Blue vs. Kasparov



Deep Blue
IBM chess computer



Garry Kasparov
World Chess Champion

- 2005 – The DARPA Grand Challenge
- A \$2 million prized race for autonomous vehicles across 100+ kms off-road terrain in the desert.



Stanford Racing Team's leader Sebastian Thrun

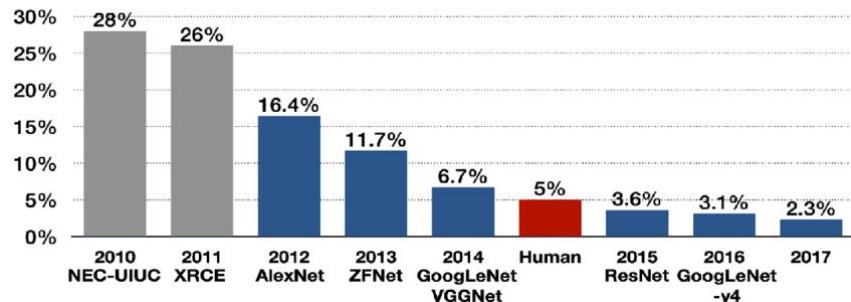
Most Amazing Milestones So Far



- 2011 – IBM Watson’s Jeopardy! Victory
- The final tally was \$77,147 to Mr. Jennings’s \$24,000 and Mr. Rutter’s \$21,600.



- 2015 – Machines “see” better than humans
- Large-scale image recognition contest for classifying 50,000 high-resolution color images into 1,000 categories.
- The model is considered to have classified a given image correctly if the target label is one of the model’s top 5 predictions.



Most Amazing Milestones So Far



- 2016 – AlphaGo created by Deep Mind (now a Google subsidiary) defeated world Go champion Lee Sedol over five matches.
- There are over 100,000 possible opening moves in Go, compared to 400 in Chess, make the brute force approach impractical.



Recent Progress



- Google Search
- Computer Vision / Image Recognition
 - ImageNet
 - Convolutional Neural Networks
- Autonomous driving
- Speech Recognition
- Voice assistants
 - Apple's Siri, Microsoft's Cortana, Amazon's Echo
- Language Translation
 - Google Translate
 - Unsupervised Translation
- Game Playing / Deep Reinforcement Learning
 - AlphaGo



ML vs DL



Traditional Machine Learning



Requires handcrafted features

Car ✓
Truck ✗
•
Bicycle ✗

Deep Learning



Convolutional Neural Network (CNN)

End-to-end learning

Feature learning + Classification

Car ✓
Truck ✗
•
Bicycle ✗

Next Class



- Learning Problems and the Empirical Risk Minimization Framework
- Loss Functions for Classification and Regression
- Evaluation Metrics for Classification



References



-
1. Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning: DeepLearning.AI





Empirical Risk Minimization



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



Learning Process



Three components

1. Generator of random vectors x , i.i.d. from a fixed but unknown distribution $P(x)$
2. A supervisor (oracle/astrologer) which returns an output vector y , for every input vector x , as per the conditional distribution $P(y/x)$, also fixed but unknown.
3. A learning machine capable of implementing a set of functions

$$f(x, w), w \in W$$

Learning Process



-
- The learning problem is to choose from the given set of functions the one which best approximates the supervisor's response.
 - The selection is based on training samples

$$(x_i, y_i); i = 1, 2, 3 \dots l$$

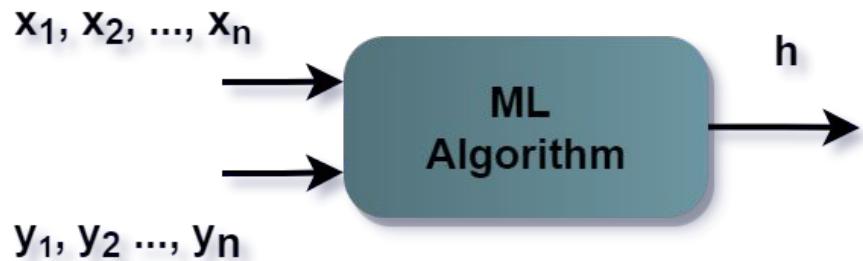


Problem Setting



- Dataset is a set of possible instances $\mathbf{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- $x_i \in \mathbf{X}$: Each sample is a vector with \mathbf{R}^d drawn from distribution $P(x)$
- $y_i \in \mathbf{Y}$:
 - Classification: Each label is a single integer value out of two [binary] or more classes [multiclass]
 - Regression: $Y = \mathbf{R}$ [real number]
- Unknown target function $f : \mathbf{X} \rightarrow \mathbf{Y}$ as distribution $P(y/x)$
- Set of function hypotheses $H = \{ h \mid h : \mathbf{X} \rightarrow \mathbf{Y} \}$
- **Input:** Training examples $\{\langle x_i, y_i \rangle\}$
- **Output:** Hypothesis $h \in H$ that best approximates target function f

Training vs Testing



Training Phase



Testing/Deployment
Phase

Sample Data



Person		height(in feet)	weight(in lbs)	foot size(in inches)
0	male	6.00	180	12
1	male	5.92	190	11
2	male	5.58	170	12
3	male	5.92	165	10
4	female	5.00	100	6
5	female	5.50	150	8

- $X = \langle \text{height, weight, foot size} \rangle$
- $Y = \langle \text{male, female} \rangle \mid \langle 0, 1 \rangle \mid \langle -1, +1 \rangle$
- A sample instance $(x^1, y^1) = (\langle 6.00, 180, 12 \rangle, \text{male})$
- Dimensionality d in $X \in \mathbb{R}^d = 3$
- Unknown target function f : height, weight, foot size \rightarrow male/female

How to choose h ?



- Randomly:
 - Advantage: Really fast
 - Disadvantage: Terrible performance
- Scan entire H and pick the best:
 - Advantage: Great performance
 - Disadvantage: Terribly slow
- Intelligent Way: Learn it using the performance metric
 - Loss functions help to evaluate the performance of the $h_i's \in H$ to identify the **optimal h** .

Loss Functions



- To choose the best function, it makes sense to minimize a loss (or cost or discrepancy) between the response of the supervisor and the learning machine, given an input (x, y)

$$\mathcal{L}(y, f(x))$$

- We want to minimize the loss over all samples

$$L(h) = \frac{1}{n} \sum_{i=1}^n l(f(x_i), y_i)$$

- Loss functions are always non-negative.
 - Hence, minimum possible loss is which means we are not making any mistakes.

Loss Functions - Classification



- **0-1 Loss:** Binary Classification with equal weights on misclassification

$$L(h) = \frac{1}{n} \sum_{i=1}^n l_{01}(f(x_i), y_i)$$
$$l_{01}(f(x_i), y_i) = \begin{cases} 0, & \text{if } f(x_i) = y_i \\ 1, & \text{if } f(x_i) \neq y_i \end{cases}$$

ID	Model Prediction $f(\mathbf{x}_i)$	Ground Truth \mathbf{y}_i	Loss $l_{01}(f(\mathbf{x}_i, \mathbf{y}_i))$
Mango 1	Good	Good	0
Mango 2	Good	Bad	1
Mango 3	Bad	Good	1
Mango 4	Bad	Bad	0
Total			0.5

Loss Functions - h_1 or h_2



- The aim of the function is to select a hypothesis with the lowest loss.

ID	Model Prediction $[h_1]$ $f(x_i)$	Model Prediction $[h_2]$ $f(x_i)$	Ground Truth y_i	Loss $l_{01}(f(x_i, y_i))$	Loss $l_{01}(f(x_i, y_i))$
M1	Good	Good	Good	0	0
M2	Good	Bad	Bad	1	0
M3	Bad	Good	Good	1	0
M4	Bad	Bad	Bad	0	0
Total Loss				0.5	0

Loss Functions - Unequal Weights



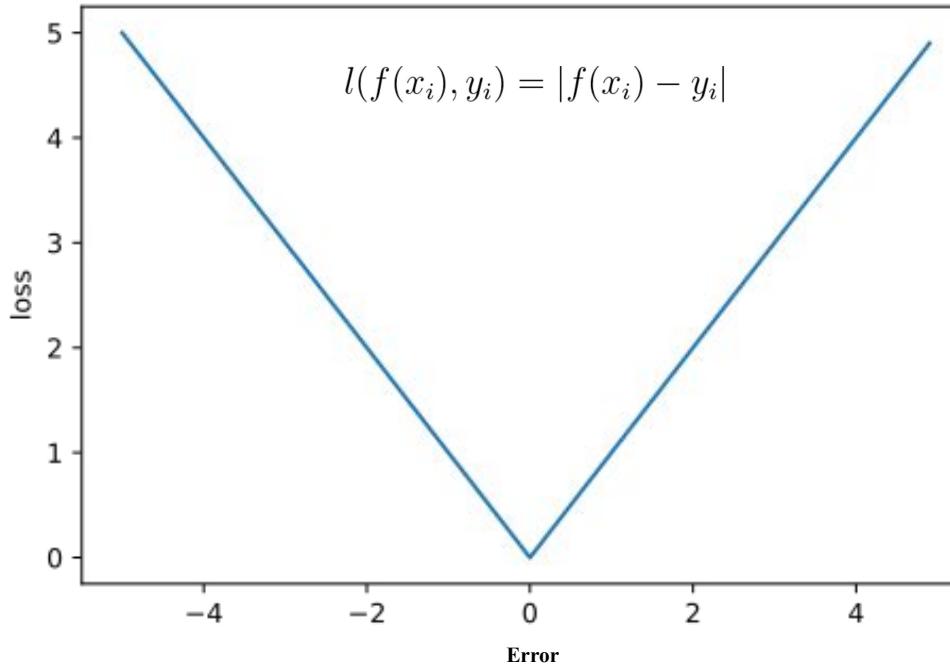
- Classification with unequal weights on misclassification
 - Minimize a 0-10⁷-500 loss

$$l(f(x_i), y_i) = \begin{cases} 0, & \text{if } f(x_i) = y_i \\ 500, & \text{if } f(x_i) = 1, y_i = 0 \\ 10^7, & \text{if } f(x_i) = 0, y_i = 1 \end{cases}$$

Loss functions: Regression - L₁ Loss

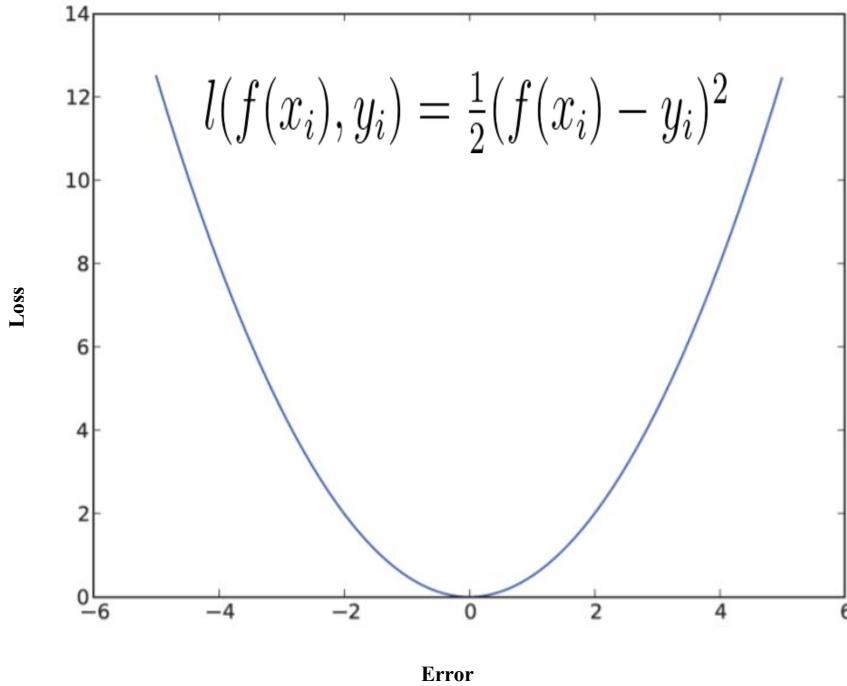


$$l(f(x_i), y_i) = |f(x_i) - y_i|$$



Loss functions: Regression - L₂ Loss

$$l(f(x_i), y_i) = \frac{1}{2}(f(x_i) - y_i)^2$$



Mean Absolute Error vs. Mean Square Error



- Mean Absolute Error (MAE)

$$L(h) = MAE = \frac{1}{n} \sum_{i=1}^n |f(x_i) - y_i|$$

- Mean Square Error (MSE)

$$L(h) = MSE = \frac{1}{n} \sum_{i=1}^n (f(x_i) - y_i)^2$$

MAE = 1, RMSE = 1.50

I D	Error	Error	Error ²
1	0	0	0
2	1	1	1
3	2	2	4
4	-0.5	0.5	0.25
5	1.5	1.5	2.25
Total		5	7.50

MSE vs. MAE (L₂ loss vs L₁ loss)



- Robust to Outliers? L₁ or L₂
 - An individual's height being too tall or small.
- Differentiability?
 - **What:** Continuous and smooth function over a region.
 - **Why Do We Care:** To get the rate of increase/decrease defined at all points.
 - **Buy Why!:** It allows to find the minima/maxima and hence the optimal model.



Generalization



- What about performance on unknown or new data samples?
- Our true goal is to know and minimize the loss of the *unknown test samples* drawn from same distribution P

$$h^* = \operatorname{argmin}_{f(x)} \frac{1}{m-n} \sum_{i=n+1}^m l(f(x_i), y_i)$$

- In other words, we want to know and minimize the *Expected Loss* $l(f(x, y))$ and hence the *Risk* associated with function f .

$$R(f) = \int \int p(x_i, y_i) l(f(x_i), y_i) dx dy$$

Expected Loss



- Usually we don't know the test points and their labels in advance!!!
 - We do not know the $p(x_i, y_i)$
 - Hence, we do not know the *expected loss* or the $R(f)$
- ***Our goal:*** *Expected loss should be closer to the actual loss*
- The law of large numbers (LLN) states that if the amount of exposure to losses increases, then the predicted loss will be closer to the actual loss.
 - Example: Insurance in Real Life

Empirical Risk Minimization Principle



- So. by LLN the *statistical risk* associated with function f becomes equal to the *empirical risk*

$$\frac{1}{m-n} \sum_{i=n+1}^m l(f(x_i), y_i) \xrightarrow{m \rightarrow \infty} R_{L,P}(f)$$

- Picking the function f (*via hypothesis h*) that minimizes the *empirical risk* is known as *empirical risk minimization*.

$$h^* = f^* = \operatorname{argmin}_{f \in F} R_{L,P}(f)$$

- **Our hope:**

$$\operatorname{argmin}_{f \in F} R_{L,P}(f) \approx \operatorname{argmin}_{f \in F} R_{L,P}^{true}(f)$$

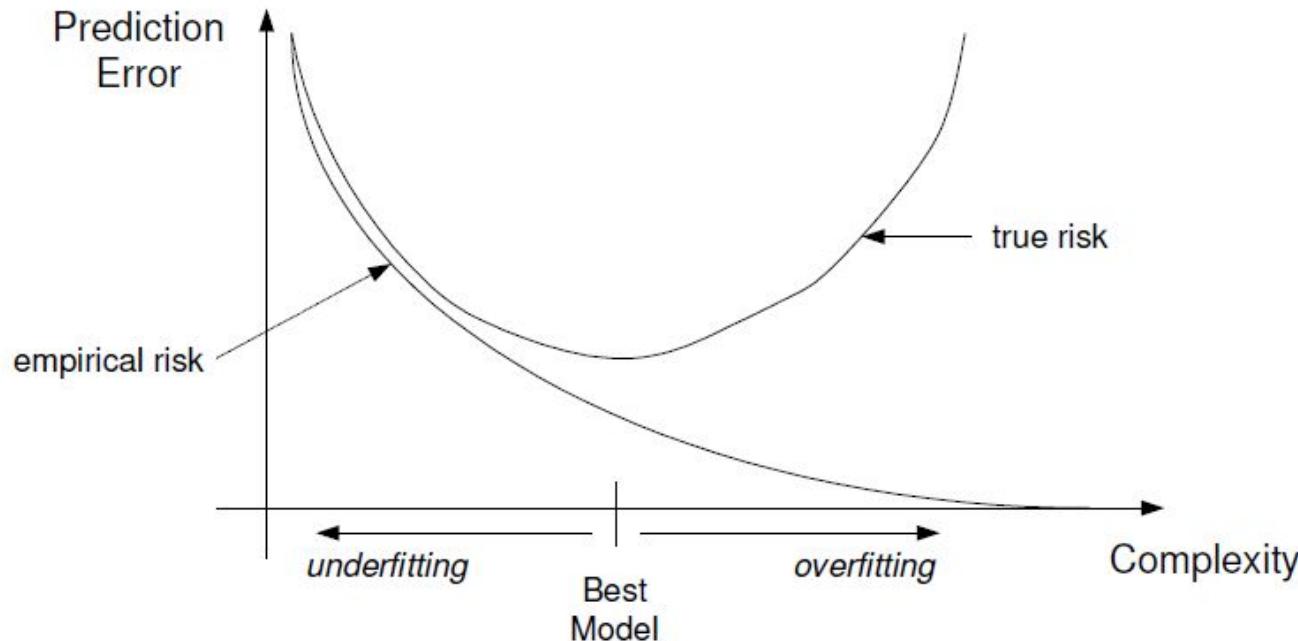
Empirical Risk Minimization Principle



Empirical risk minimization depends on following:

1. **How much data we have:** For any given function f , as we get more and more data, we can expect that $R(f) \rightarrow R_{\text{true}}(f)$
2. **The true distribution p:** Depending on how “complex” the true distribution is, more or less data may be necessary to get a good approximation of it.
3. **The loss function L:** If the loss function is very “weird”—giving extremely high loss in certain unlikely situations, this can lead to trouble.
4. **The class of functions F:** Roughly speaking, if the size of F is “large”, and the functions in F are “complex”, this worsens the approximation, all else being equal.

Effect of Function Complexity



- At fixed number of samples, overly complicated models -> overfitting.
- Empirical risk is no longer a good indicator of true risk.



References



-
- [Principles of Risk Minimization for Learning Theory](#)



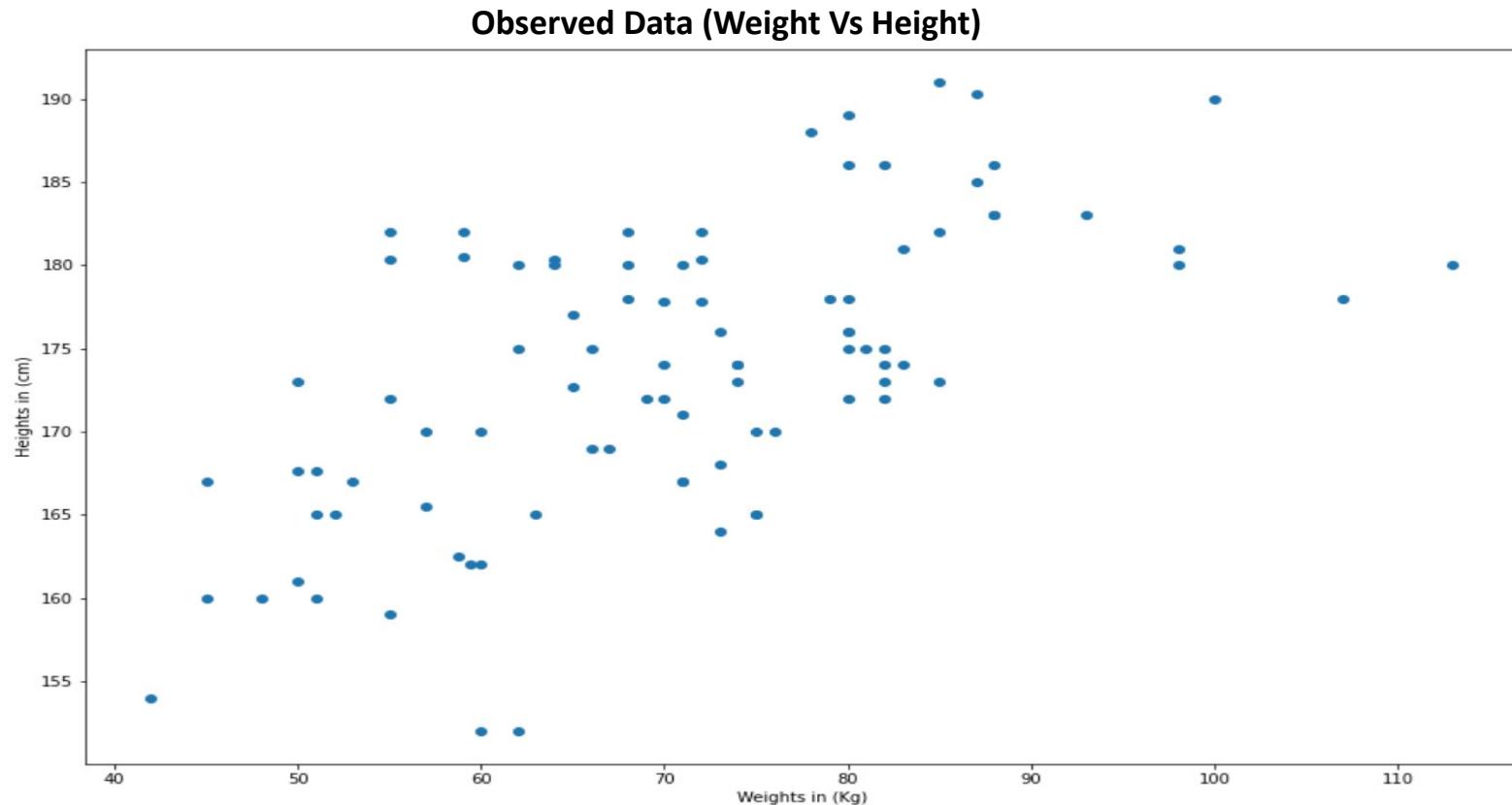
Linear Models for Regression



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



The Problem



Motivation



- Height and weight are random.
- Can we accurately predict what will be someone's height given her weight?
 - Difficult to estimate from “a priori” models
 - But, we have lots of data from which to build a model
- Assumptions:
 - Linear correspondence (relationship) exists between height and weight.
 - Usually true
 - Observation noise follows a normal distribution.
 - Also usually true!

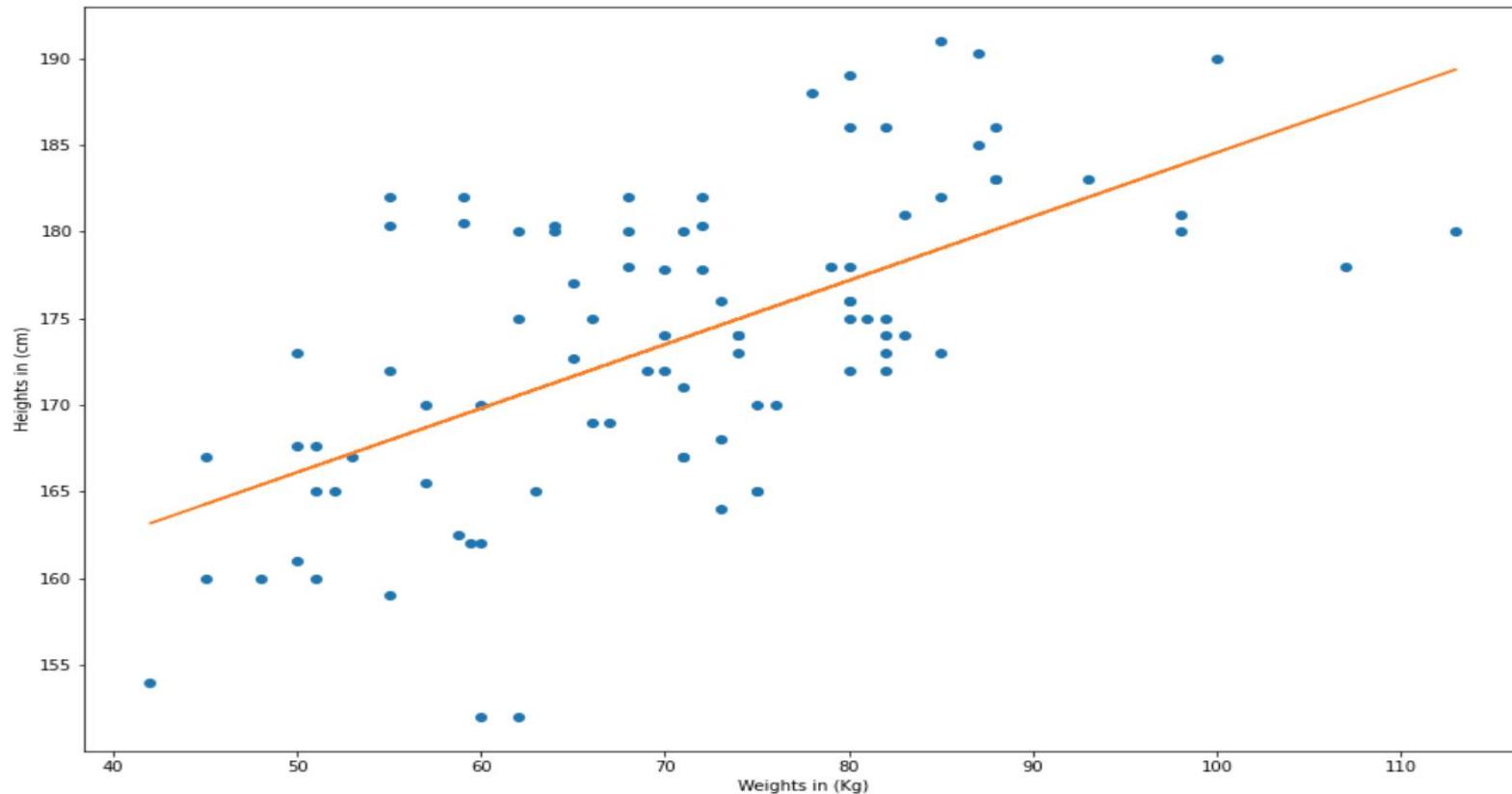
Formal Problem Settings



- **Input:** $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- $x_i \in X, y_i \in Y$ where $X, Y \subseteq \mathbb{R}$ [real number]
- Hypothesis $y_i = w^T x_i + \varepsilon$
 - Linear Correspondence -> $w^T x_i$
 - Normal Distribution of noise -> $\varepsilon \sim N(0, \sigma^2)$
- $y_i \sim N(w^T x_i, \sigma^2)$

$$P(y_i | \vec{x}_i; w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(\frac{w^T x_i - y_i}{2\sigma}\right)^2}$$

A simple model



How to get W?



- **Maximum Likelihood Estimation (MLE)** gives us the solution which maximises the likelihood.
 - Find w that maximizes the probability of the data D
 - $\operatorname{argmax} P(D|w)$
- **Maximum A Posterior (MAP)** gives us the solution which maximises the posterior probability.
 - Find w that is most likely given the data D .
 - $P(w|D) = P(D|w) * P(w)/P(D)$
 - Assumes the availability of the prior $P(w) \sim N(\mu, \sigma^2)$
 - E.g. in case of transfer learning as initial weights w_o

How to get W: Maximum Likelihood Estimation (MLE)

- MLE gives us the solution which maximises the Likelihood

$$\operatorname{argmax}_w \prod_{i=1}^n P(y_i | \vec{x}_i; w) = \operatorname{argmax}_w \sum_{i=1}^n \log P(y_i | \vec{x}_i; w)$$

- Both will provide the same maxima. Now putting the value of the normal distribution probability:

$$= \operatorname{argmax}_w \sum_{i=1}^n \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{2\sigma^2}(w^T \vec{x}_i - y_i)^2$$

- The \log term is independent of w , hence we can get rid of it

Finding model parameters [Optimization]



- Simplifying it further:

$$= \underset{w}{\operatorname{argmax}} - \frac{1}{2\sigma^2} \sum_{i=1}^n (w^T x_i - y_i)^2$$

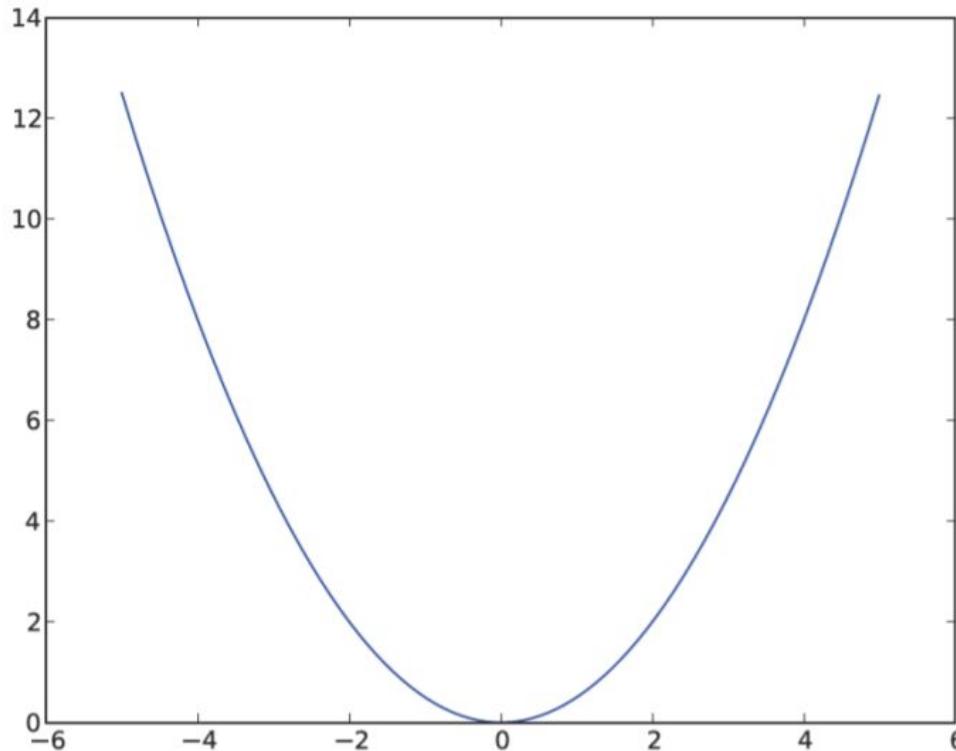
- $1/2\sigma^2$ is again a constant. Further, negative of maximization is same as minimization, hence:

$$= \underset{w}{\operatorname{argmin}} \sum_{i=1}^n (w^T x_i - y_i)^2$$

- This is the L₂ loss. To add interpretability to the above, we need to take an average:

$$J(w) = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (w^T x_i - y_i)^2$$

Loss functions: Squared Loss (L_2 Loss)



$$l(\hat{y}, y_i) = (\hat{y} - y_i)^2 = (w^T x_i - y_i)^2$$

Analytical Solution



- In the multivariate case:

$$W = (X'X)^{-1} X'Y$$

- Exercise 1: Find the solution for unidimensional case:

Hint: $J(w) = \sum_{i=1}^n (w_0 + w_1 x_i - y_i)^2$

$$\frac{\partial}{\partial w_0} = ?$$

$$\frac{\partial}{\partial w_1} = ?$$

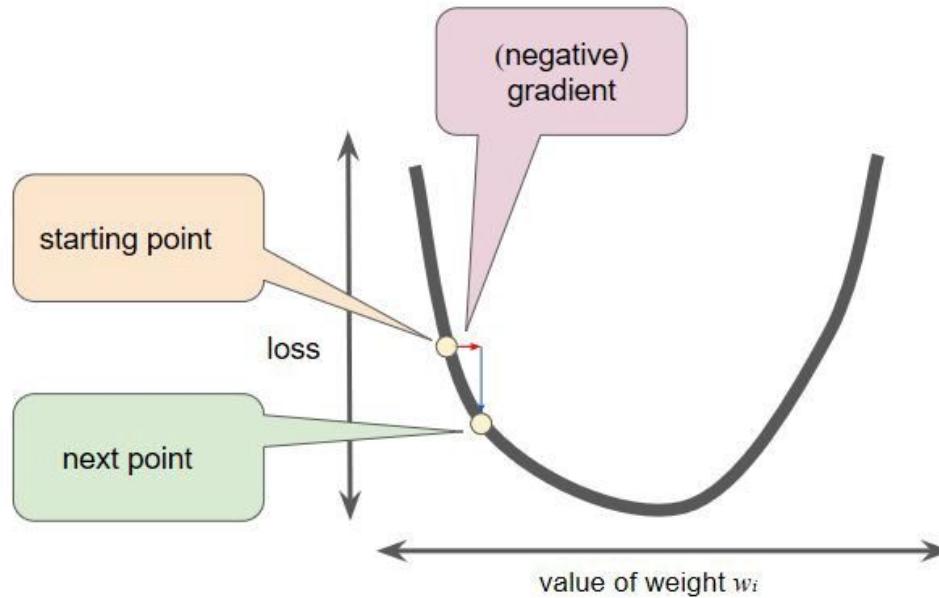
- For a multivariate case, the computational complexity is very high!
 - Hence, an iterative solution such as gradient descent is preferred.

Gradient descent



$$w_i = w_i - \eta \frac{\partial J(w)}{\partial w_i}$$

- Repeat until “convergence”



Gradient descent: Demo



-
- <https://lukaszkujawa.github.io/gradient-descent.html>

How to get W : Maximum A Posterior Estimation (MAP)

- Study the Maximum A Posteriori (MAP) solution for Linear Regression





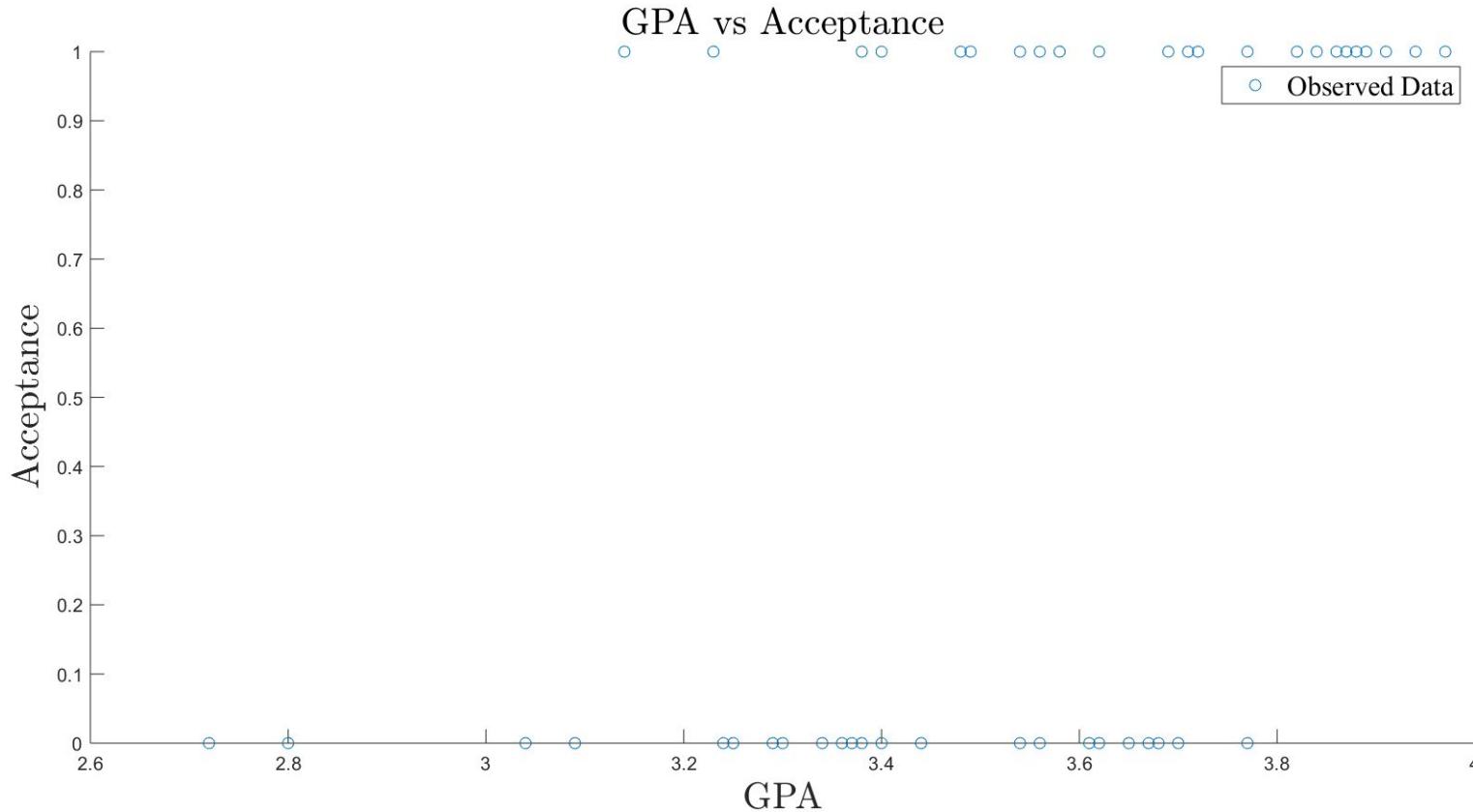
Logistic Regression



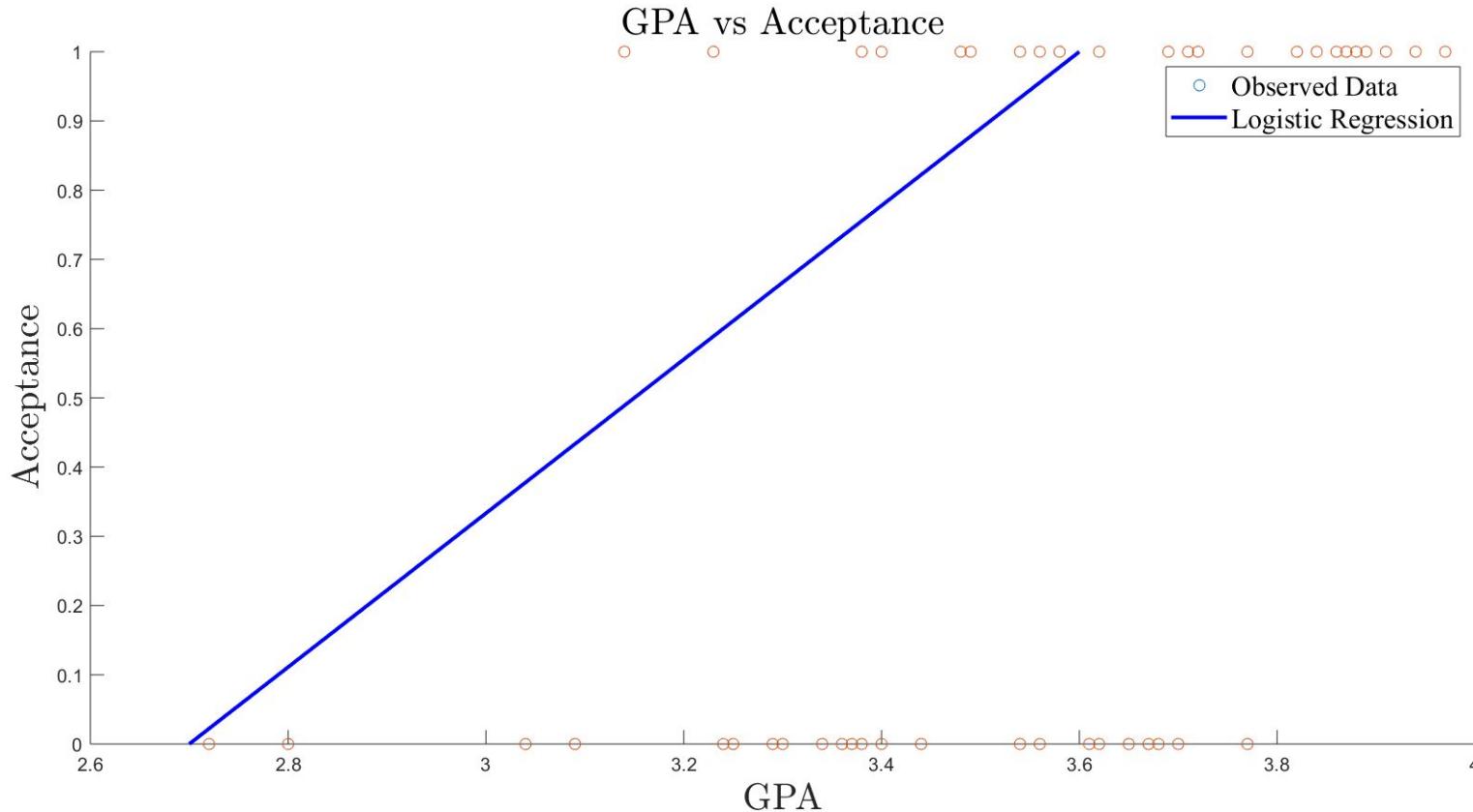
INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



The Problem



The Problem



Motivation



-
- Logistic regression is the type of regression we use for a binary (or discrete) response variable ($Y \in \{0,1\}$)
 - Linear regression is the type of regression we use for a continuous, normally distributed response ($Y \in \square^m$) variable
 - Use a function to map real numbers to $\{0,1\}$



Dependent Variable Characteristics



- Each trial has two possible outcomes: success or failure.
- The probability of success (call it p) is the same for each trial.
- The trials are independent, meaning the outcome of one trial doesn't influence the outcome of any other trial.

Bernoulli Distribution



$$\begin{aligned} Pr(y|x; p) &= \begin{cases} p, & y = 1 \\ 1 - p, & y = 0 \end{cases} \\ &= p^y(1 - p)^{(1-y)} \end{aligned}$$

- Input: Linear combination of variables
- Output: Bernoulli distribution p

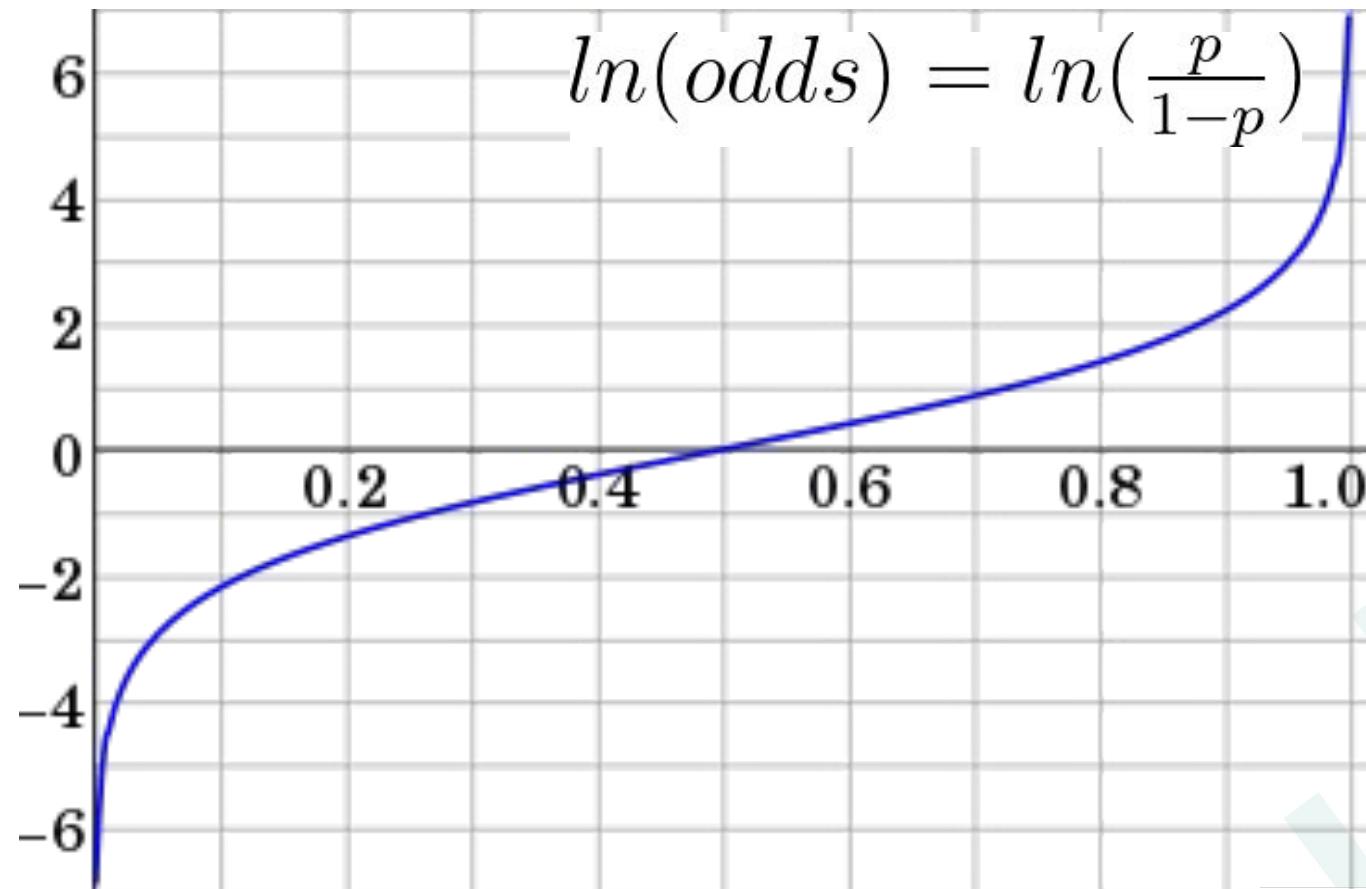
Logit



$$\ln(\text{odds}) = \ln\left(\frac{p}{1-p}\right) = \ln(p) - \ln(1-p)$$

- Range = -inf to +inf
 - Solves the problem we encountered in fitting a linear model to probabilities
 - P only range from 0 to 1, we can get linear predictions that are outside of this range

Logit



Inverse Logit



- We want to predict p , hence p has to be our Y axis rather X axis.
- The inverse of the logit function is the sigmoid function
- $\text{logit}^{-1}(z) = \sigma(z) = 1/(1 + \exp(-z))$

Inverse Logit: Derivation



$$\text{logit}(p) = \log \frac{p}{1-p}$$

Let $\text{logit}(p) = \hat{y}$:

$$\hat{y} = \log \frac{p}{1-p}$$

Taking exponential both the sides:

$$e^{\hat{y}} = \frac{p}{1-p}$$

$$e^{\hat{y}} + 1 = \frac{p}{1-p} + 1$$

Adding 1 both the sides:

$$e^{\hat{y}} + 1 = \frac{1}{1-p}$$

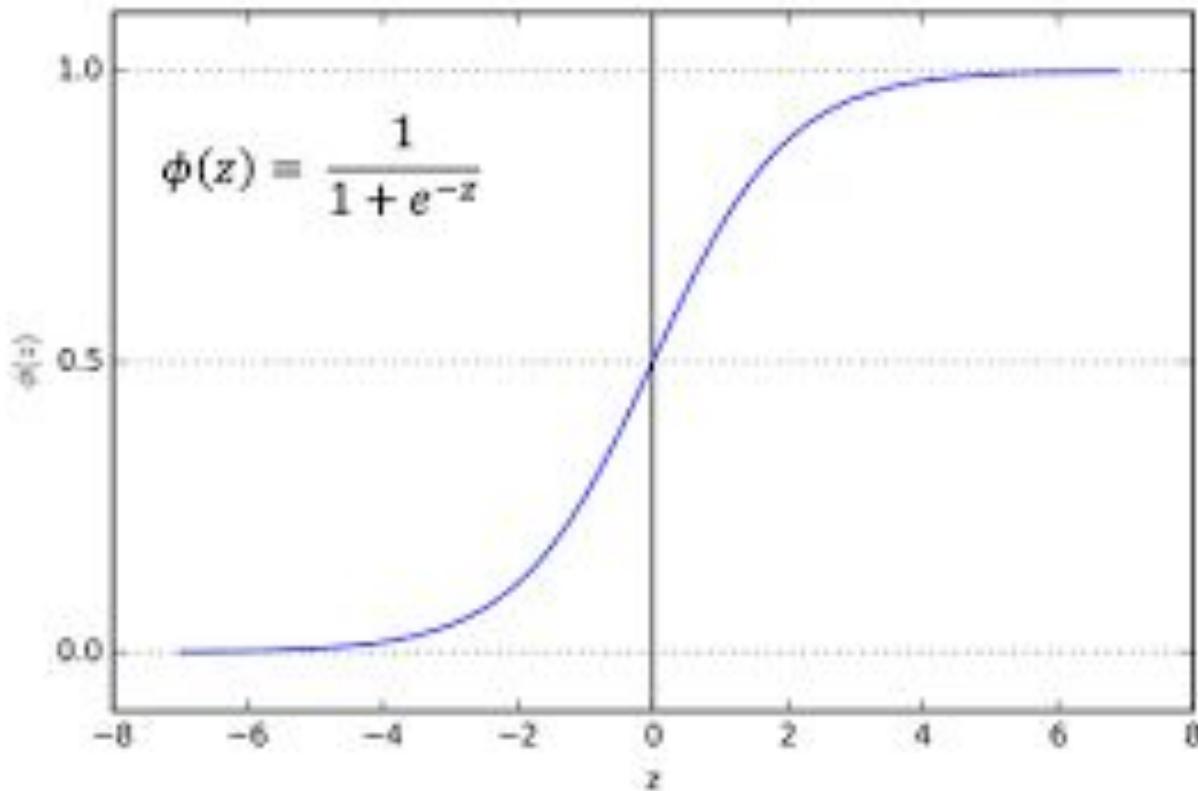
Cross-Multiplication:

$$1 - p = \frac{1}{e^{\hat{y}} + 1}$$

Simplifying it further:

$$p = \frac{e^{\hat{y}}}{e^{\hat{y}} + 1} = \frac{1}{1 + e^{-\hat{y}}}$$

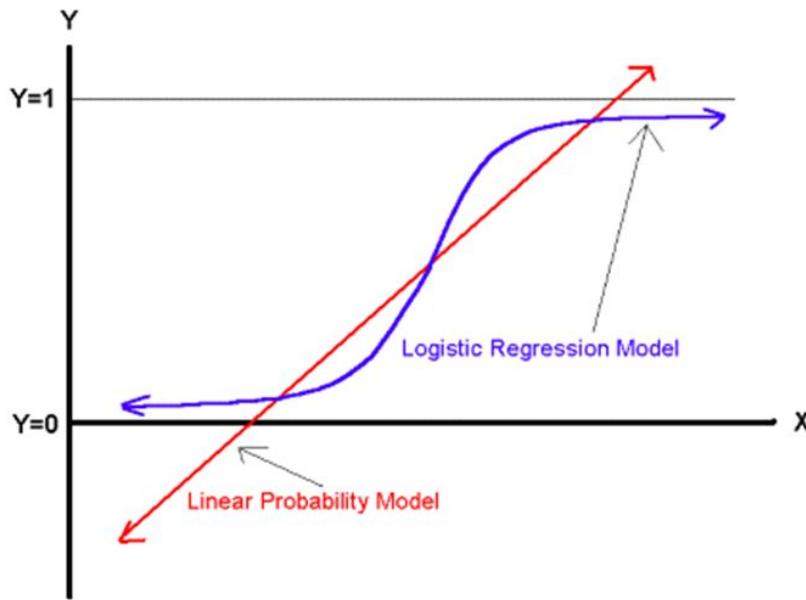
Inverse Logit



Linear Regression vs Logistic



Comparing the LP and Logit Models



Linear Regression vs Logistics Regression



- Linear regression we had
 - $h_{\theta}(x) = \theta^T x$
- logistic regression we have
 - $h_{\theta}(x) = 1/(1+e^{-\theta^T x})$



Parameter Estimation: Linear Regression on 'log(odds)'



- Hypothesis
(Predicted)

$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

- Cost Function

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2.$$

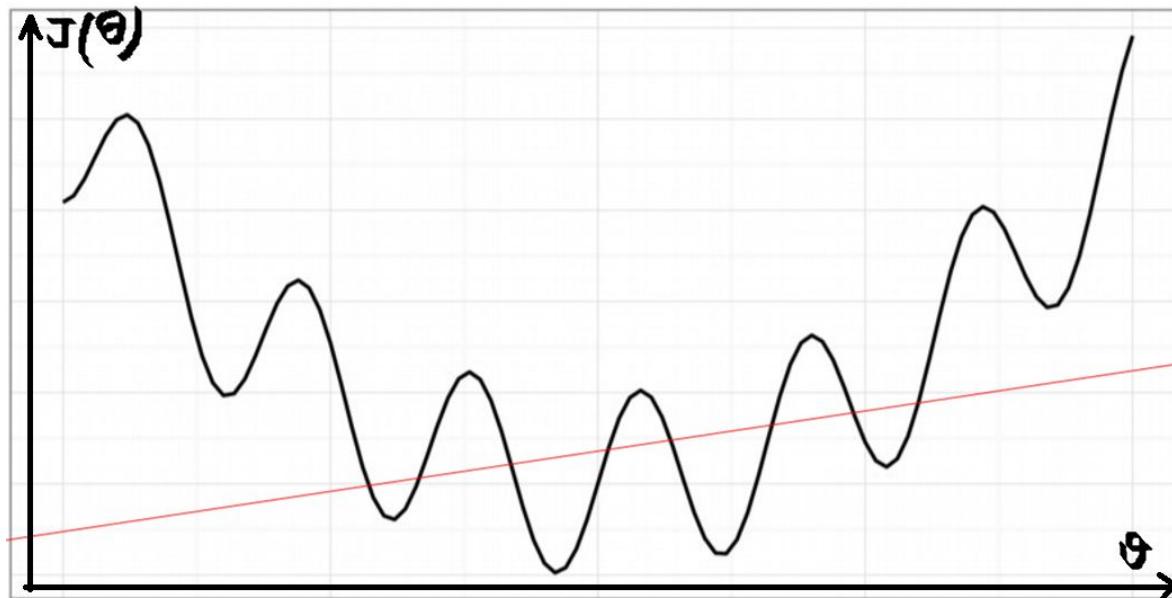


Parameter Estimation: Linear Regression on 'log(odds)'



- Hypothesis

$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$



Parameter Estimation: Linear Regression on 'log(odds)'



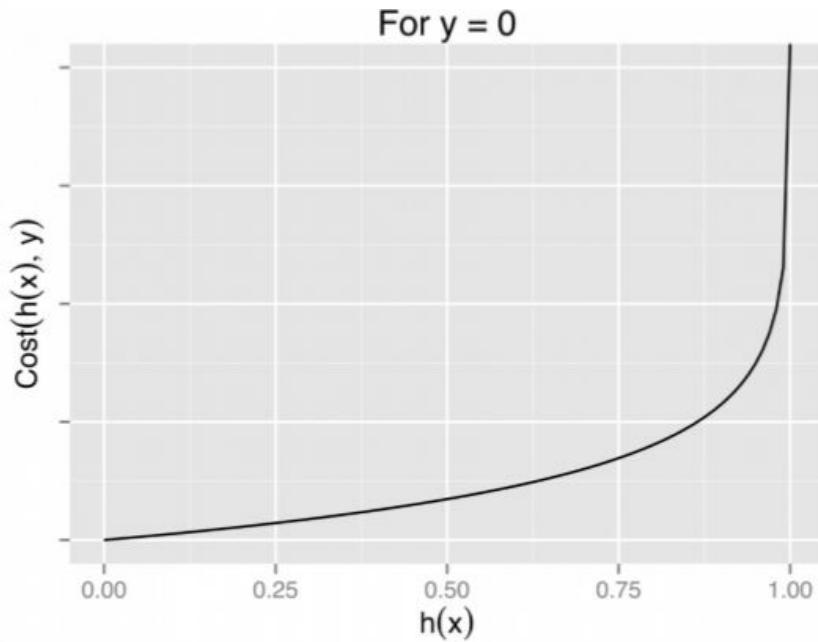
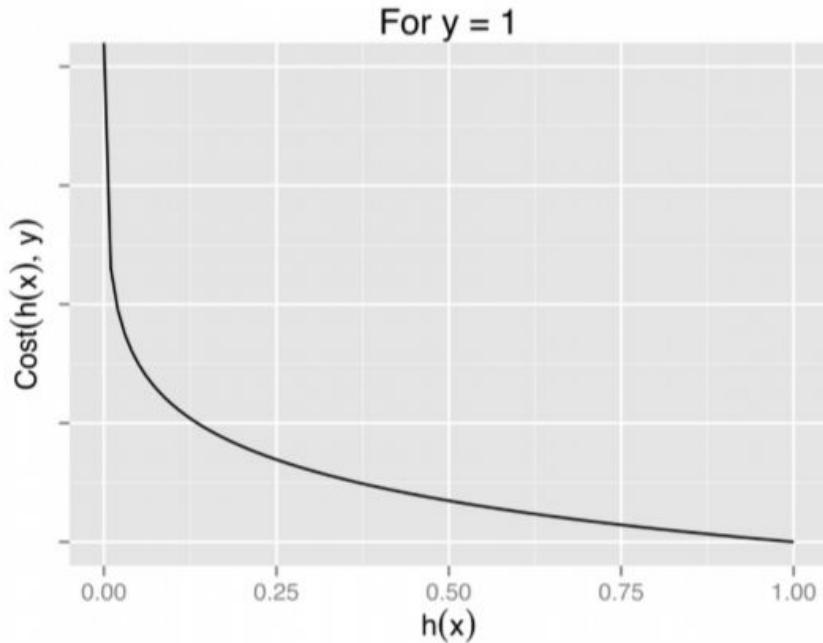
- Hypothesis

$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

- Cost Function

$$\begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Parameter Estimation: Linear Regression on 'log(odds)'



General Cost Function



$$J(\theta) = -\frac{1}{m} \sum \left[y^{(i)} \log(h\theta(x(i))) + (1 - y^{(i)}) \log(1 - h\theta(x(i))) \right]$$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

(simultaneously update all θ_j)



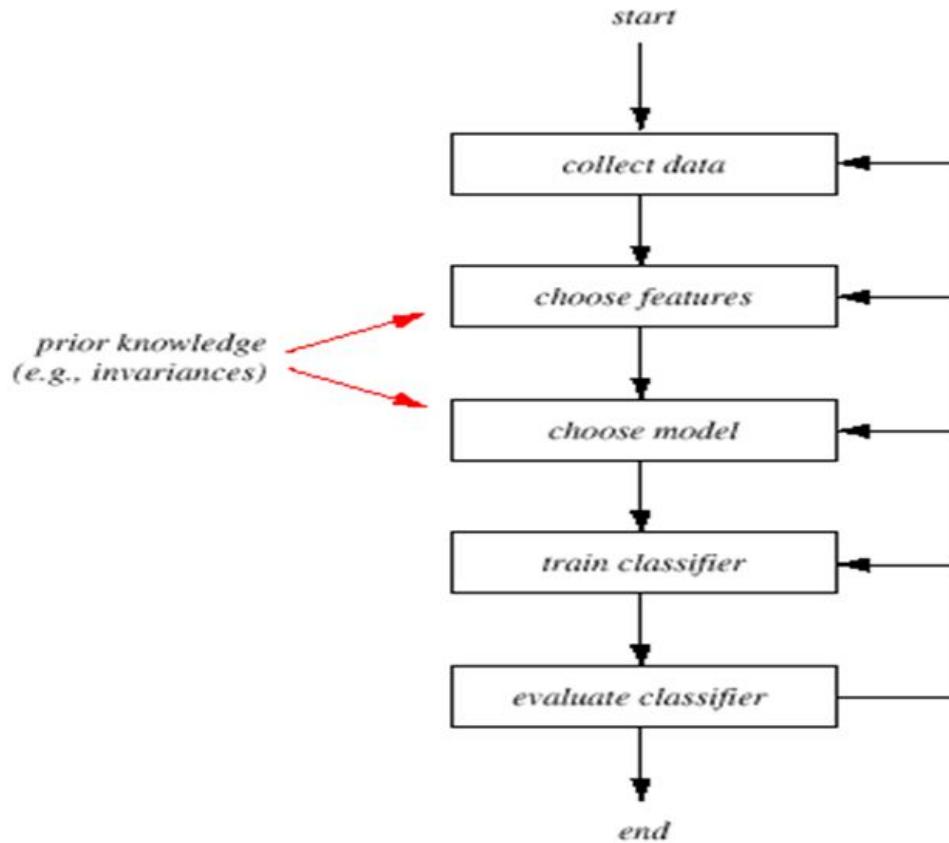
Machine Learning in Practise



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



The Design Cycle



Computational Complexity



- What is the trade-off between computational ease and performance?
- How an algorithm scales as a function of the number of features, patterns or categories?

Performance Evaluation of Learning Tasks

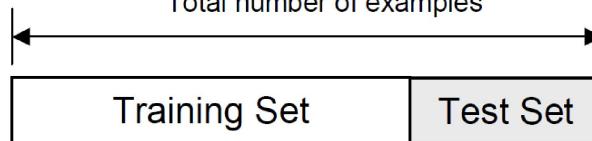


- Entire population is unavailable
- Finite set of training data, usually smaller than desired
- Naïve approach: use all available data
 - The final model will typically **overfit** the training data
 - More pronounced with high-capacity models (e.g., neural nets)
 - The true error rate is **underestimated**
 - Not uncommon to have 100% accuracy on training data

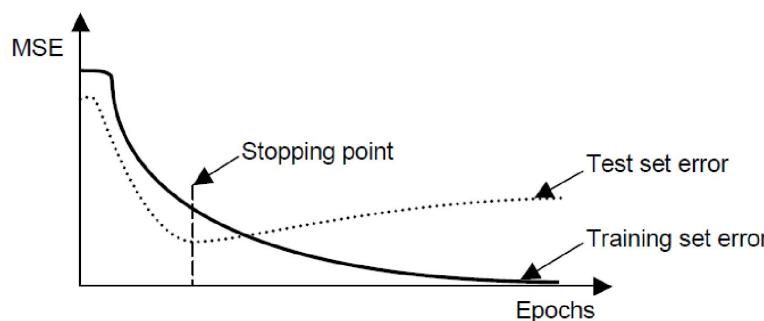
Validation Method: Holdout



- Split dataset into two groups:
 - Training set: used to train the model
 - Test set: used to estimate the error rate of the trained model



- Typical application: early stopping



Holdout



- Drawbacks

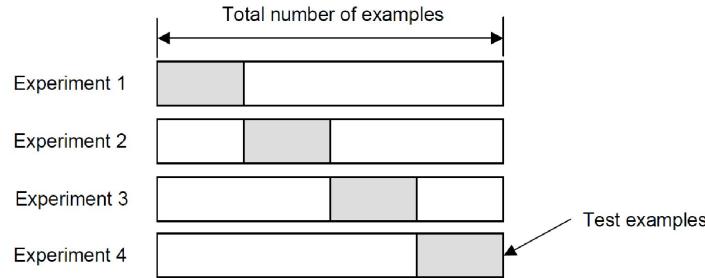
- For small training sets, setting aside a subset may be infeasible
 - Sample Size = 10 => Training set = 7, Testing set = 3
- For a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an '*unfortunate*' split
 - Training Set = 1,0,1,0,0,0,0, Testing set = 1,1,1

- Alternatives: a family of resampling methods: Cross Validation

- Random Subsampling
- Leave-one-out Cross-Validation
- K-Fold Cross-Validation

Validation Method: K-Fold Cross-validation

- Create a K-fold partition of the dataset
 - For each of K experiments, use K-1 folds for training and the remaining one for testing



- True error is estimated as the average error rate

$$E = \frac{1}{K} \sum_{i=1}^K E_i$$

Validation Method: K-Fold Cross-validation

- Example: [0.1, 0.2, 0.3, 0.4, 0.5]
- K = 3

Fold	Training Set	Testing Set
1	[0.1, 0.2, 0.3]	[0.4, 0.5]
2	[0.1, 0.2, 0.5]	[0.3, 0.4]
3	[0.1, 0.4, 0.5]	[0.2, 0.3]

Validation Method: K-Fold Cross-validation

- Example: [0.1, 0.2, 0.3, 0.4, 0.5]
- K = 3

Fold	Training Set	Testing Set	Predictions
1	[0.1, 0.2, 0.3]	[0.4, 0.5]	[0.3, 0.5]
2	[0.1, 0.2, 0.5]	[0.3, 0.4]	[0.35, 0.35]
3	[0.1, 0.4, 0.5]	[0.2, 0.3]	[0.23, 0.33]

Calculate average error rate

Validation Method: K-Fold Cross-validation

Fold	Testing Set	Predictions	Error Rate
1	[0.4, 0.5]	[0.3, 0.5]	0.005
2	[0.3, 0.4]	[0.35, 0.35]	0.0025
3	[0.2, 0.3]	[0.23, 0.37]	0.0058

Average Error Rate= 0.0044

How many folds are needed?



- Large number of folds
 - + smaller bias of the true error rate estimator
 - - larger variance of the true error rate estimator
 - - higher computational time (many experiments)
- Small number of folds
 - + lower computation time
 - + smaller variance
 - - larger bias
- In practice, the choice of the number of folds depends on the size of the dataset
 - For large datasets, even 3-Fold Cross Validation is reasonable
 - For very sparse datasets, '*leave-one-out*' is beneficial
- A common choice for K-Fold Cross Validation is K=10

Bias and Variance



Two ways to measure the “match” or “alignment” of the learning algorithm.

- Bias measures accuracy of the match: high => poor match
 - Bias arises when the classifier cannot represent the true function – that is, the classifier underfits the data
- Variance measures precision of match: high => weak match
 - Variance arises when the classifier overfits the data.
- There is often a tradeoff between bias and variance.

Bias and Variance



Calculate bias, variance and give conclusions.

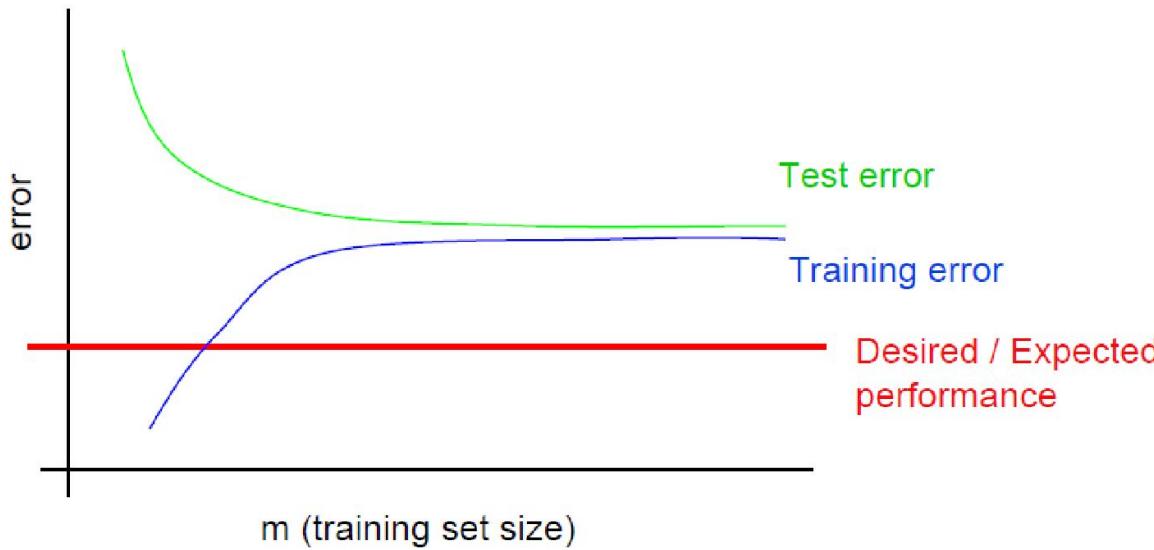
Training error	Dev error	Bias	Variance	Conclusions
1%	11%			
15%	16%			
15%	30%			
0.5%	1%			

Bias and Variance



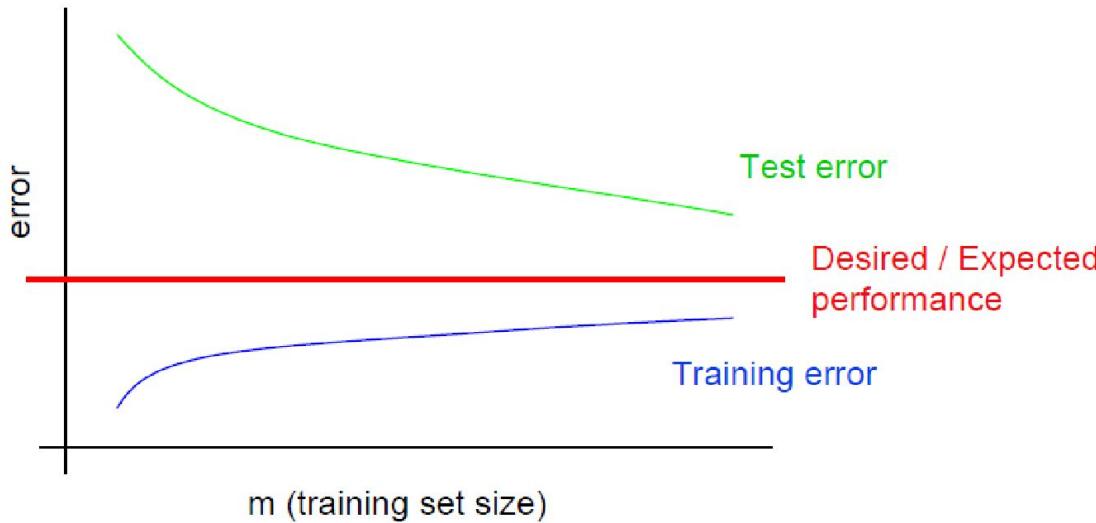
Training error	Dev error	Bias(Training error)	Variance(Dev error - Training error)	Conclusions
1%	11%	1%	10%	High variance, overfitting
15%	16%	15%	1%	High bias, underfitting
15%	30%	15%	15%	High bias and high variance, poor performance
0.5%	1%	0.5%	0.5%	Low bias and low variance, good performance

Bias vs. Variance Analysis: High Bias



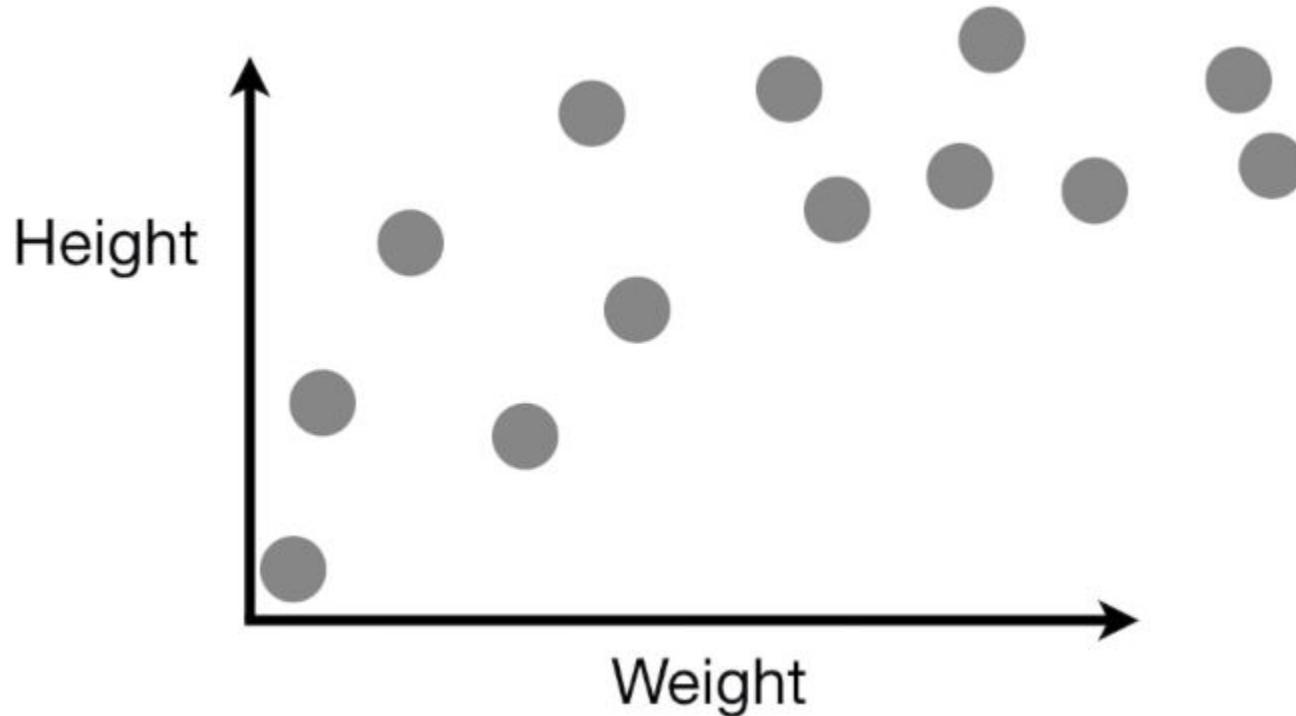
- Even training error is unacceptably high.
 - Features are not discriminative enough
- Small gap between training and test error.
 - Likely underfitting: a higher capacity model could be tried

Bias vs. Variance Analysis: High Variance

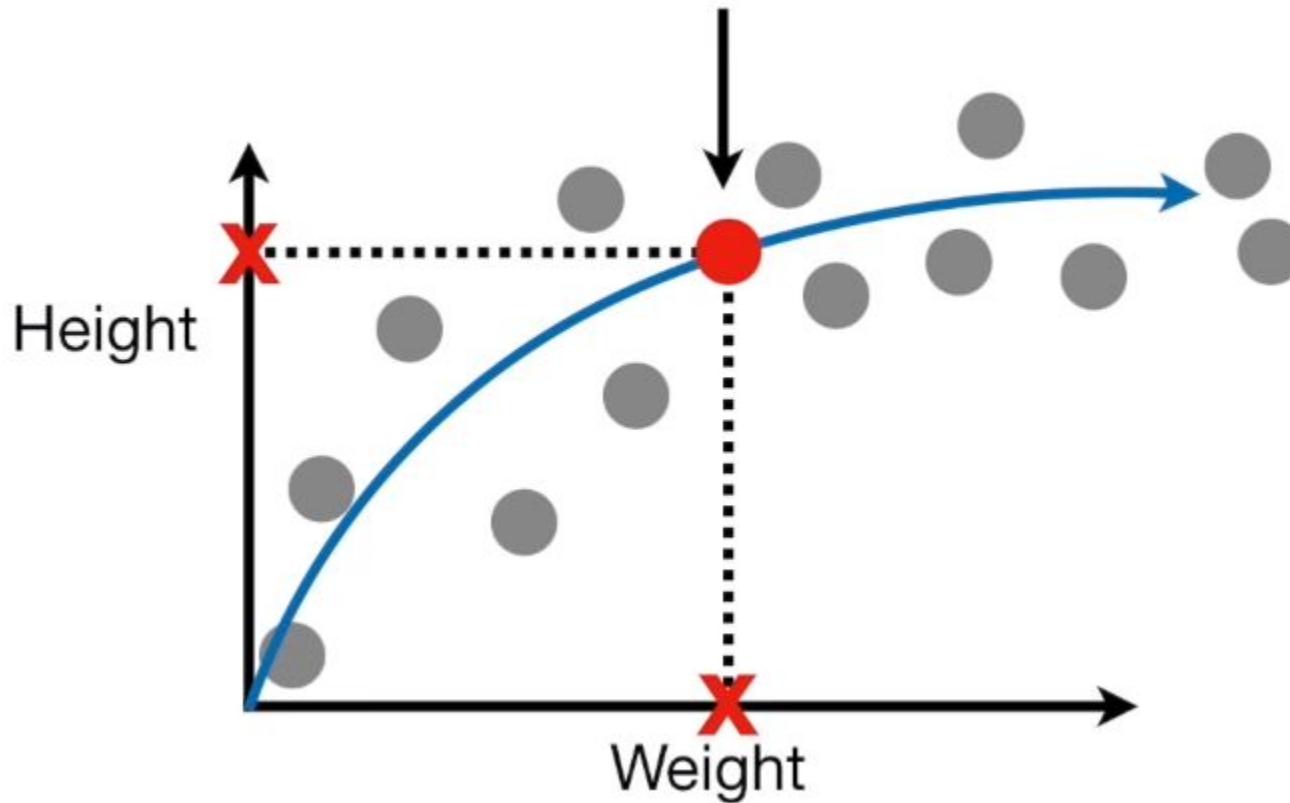


- Test error still decreasing as training set size increases.
 - Suggests a larger training set will help.
- Large gap between training and test error
 - Likely overfitting

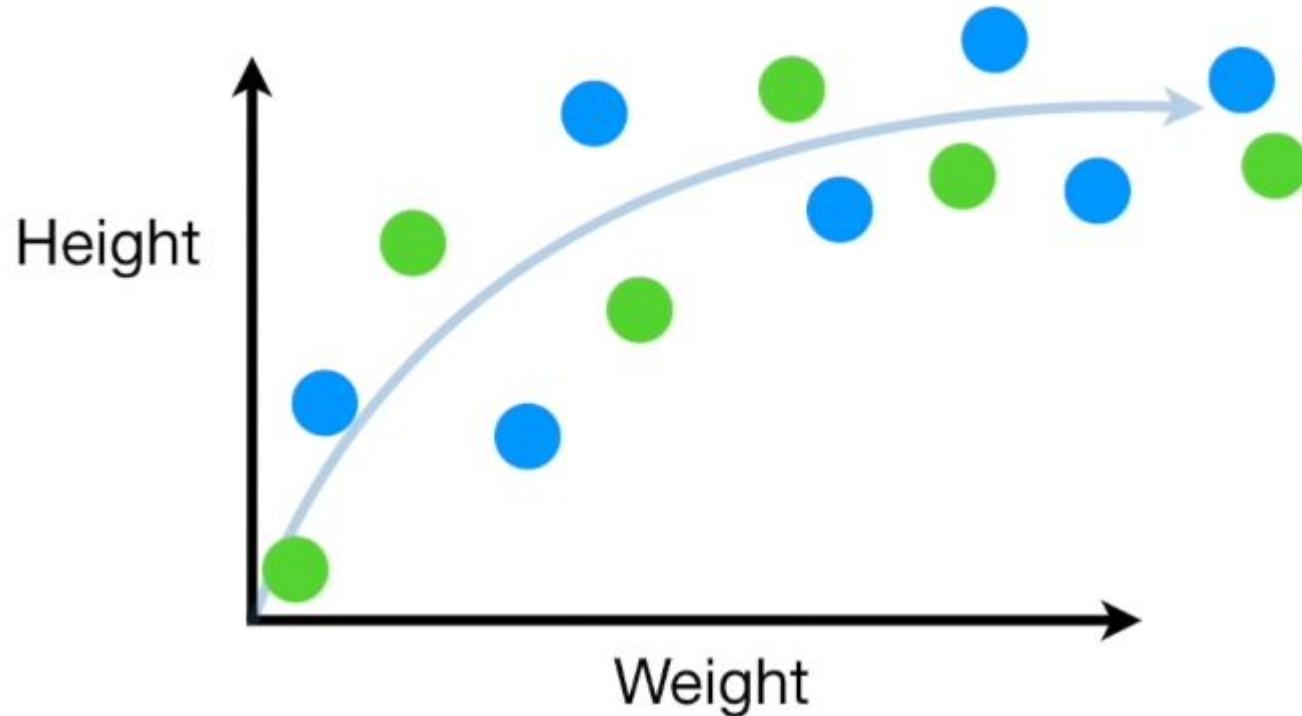
A sample data [Regression]



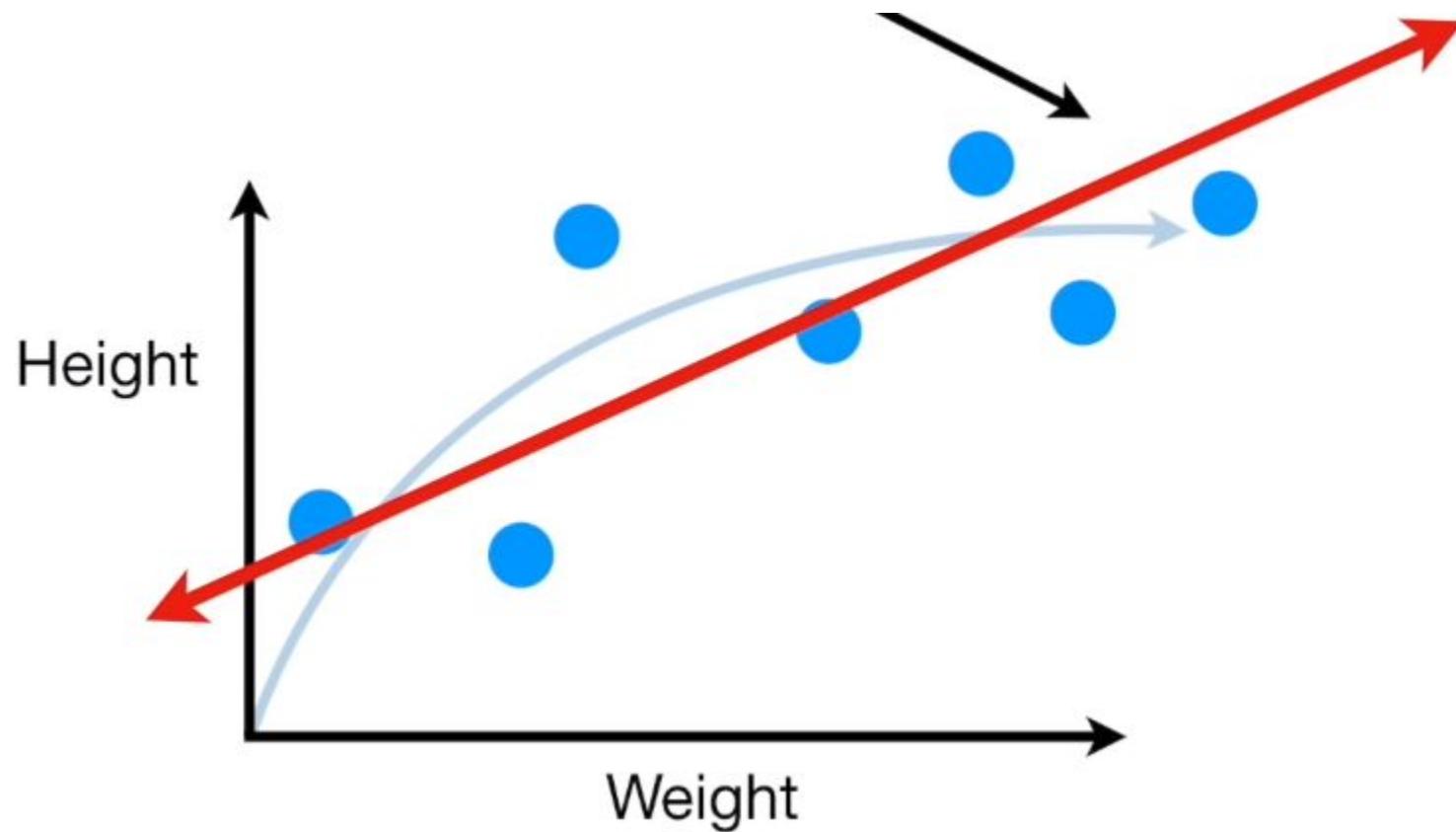
True Relationship



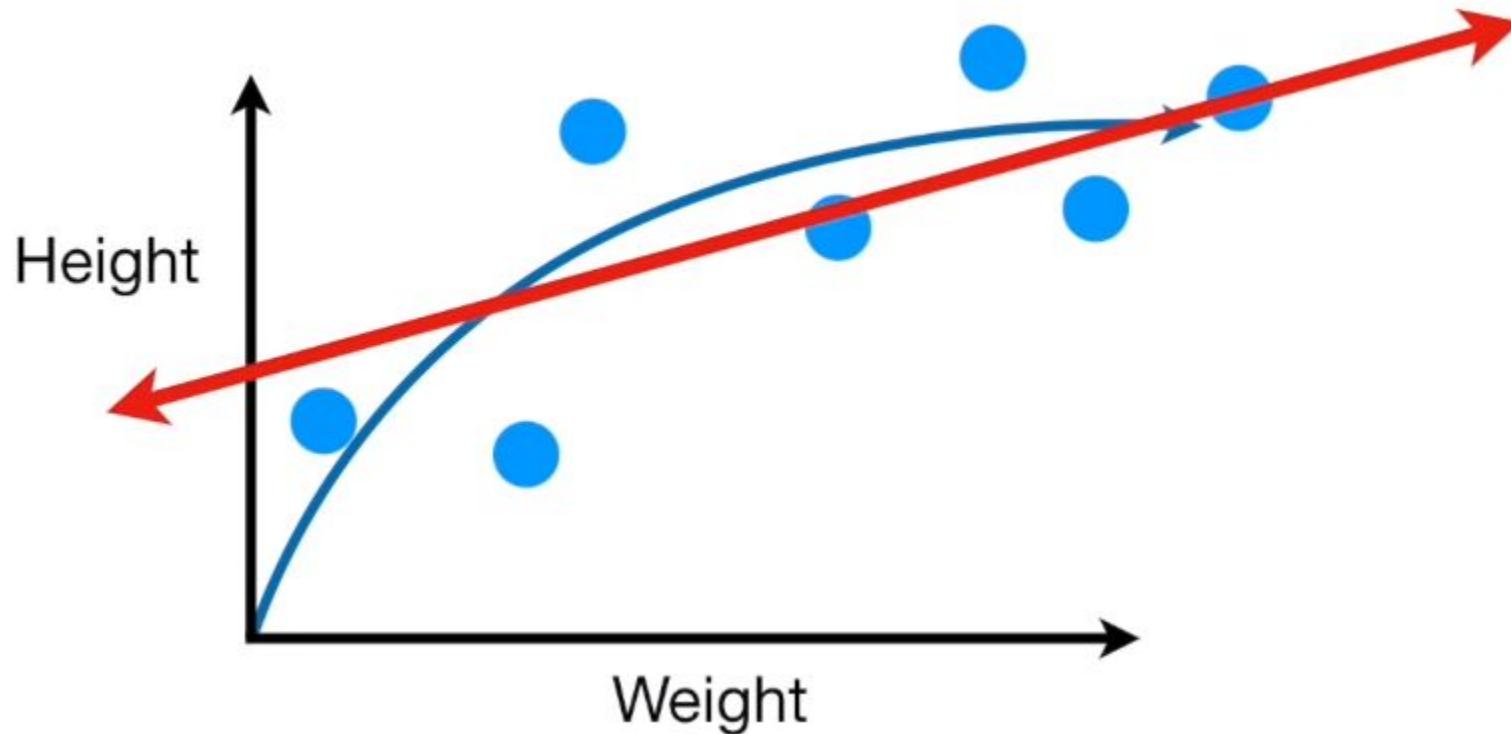
Training and Testing Data



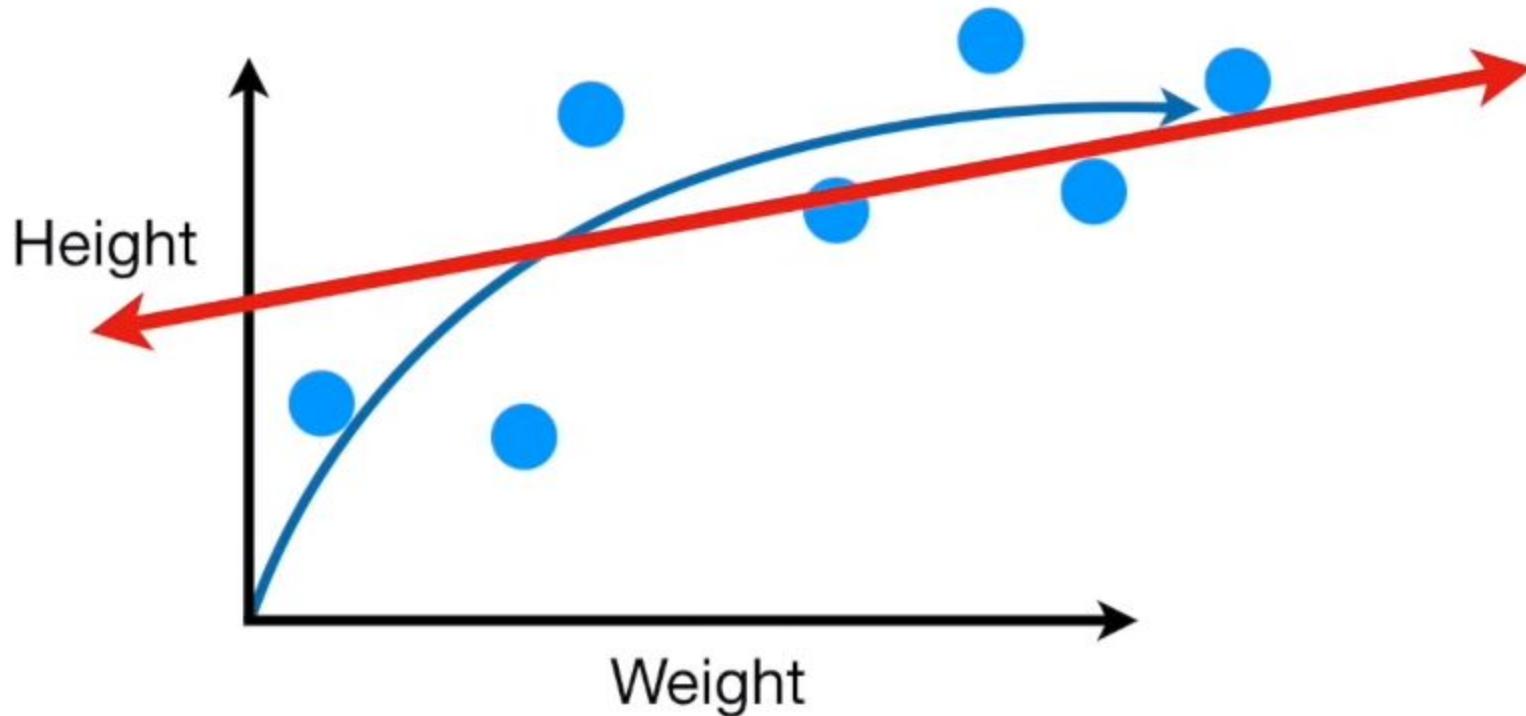
Linear Regression Fit - I



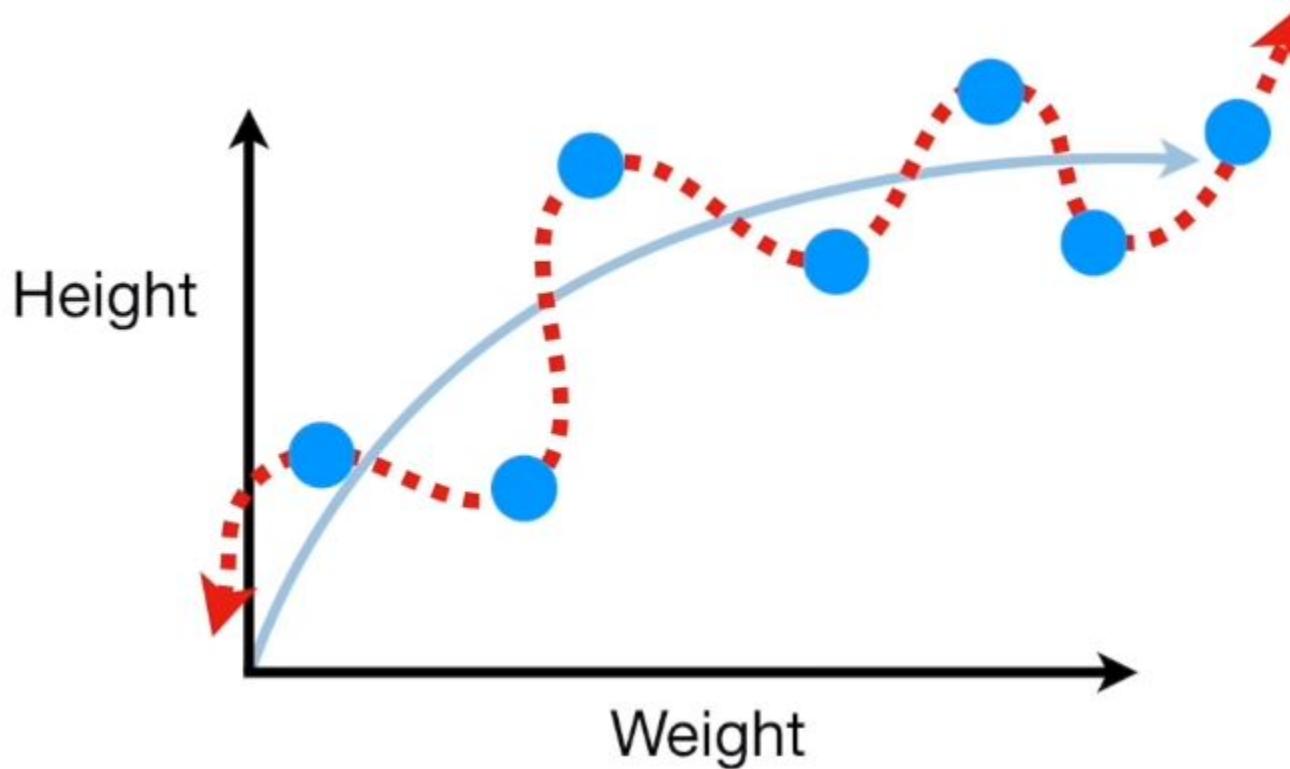
Linear Regression Fit - II



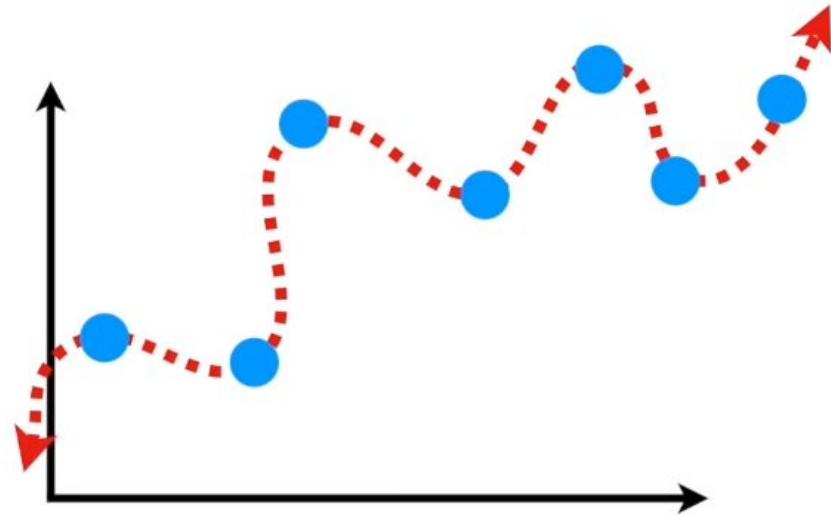
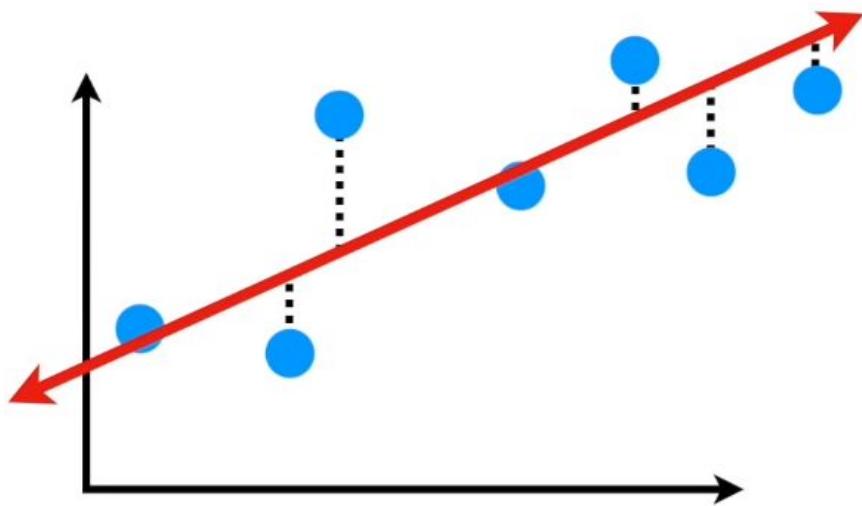
Linear Regression Fit - III



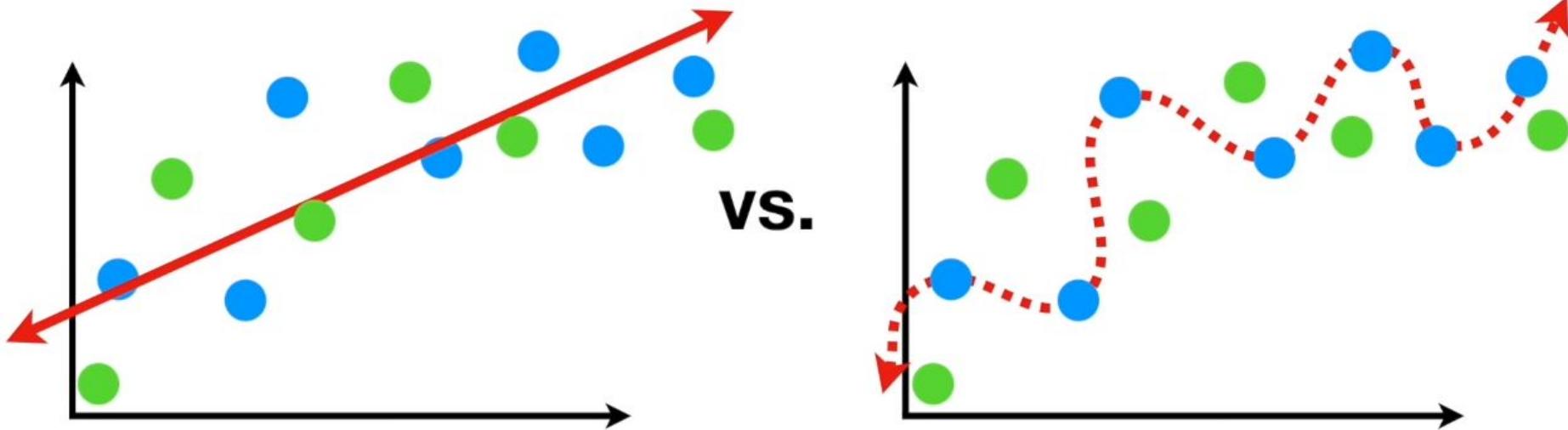
Polynomial Fit



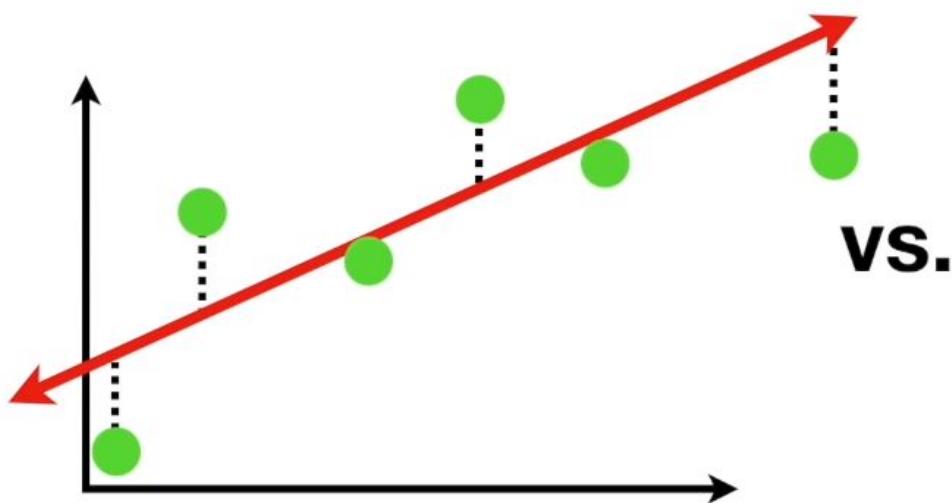
Bias



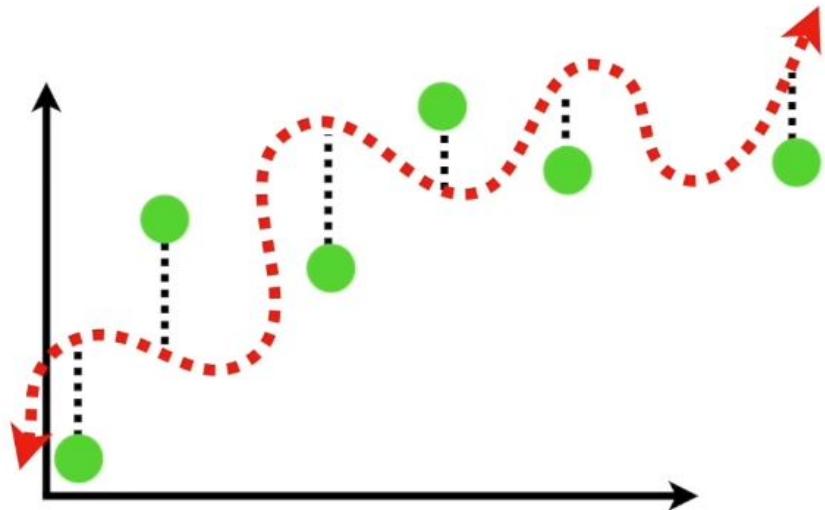
Performance on Testing Data



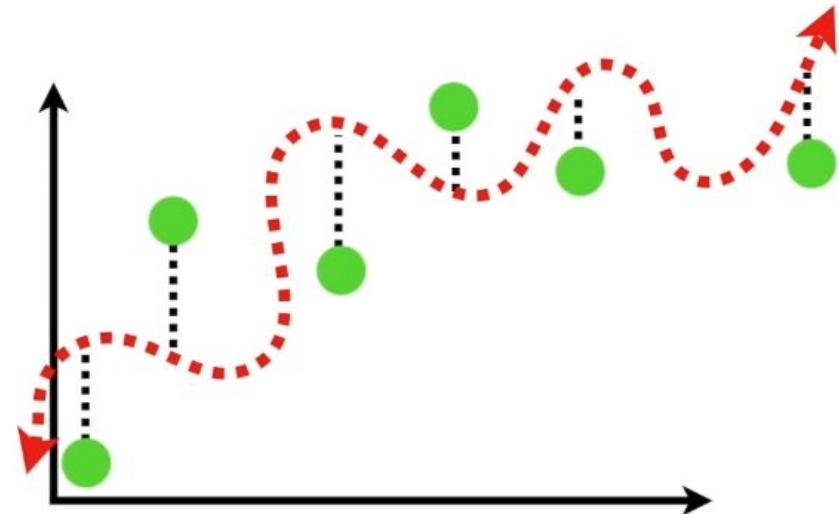
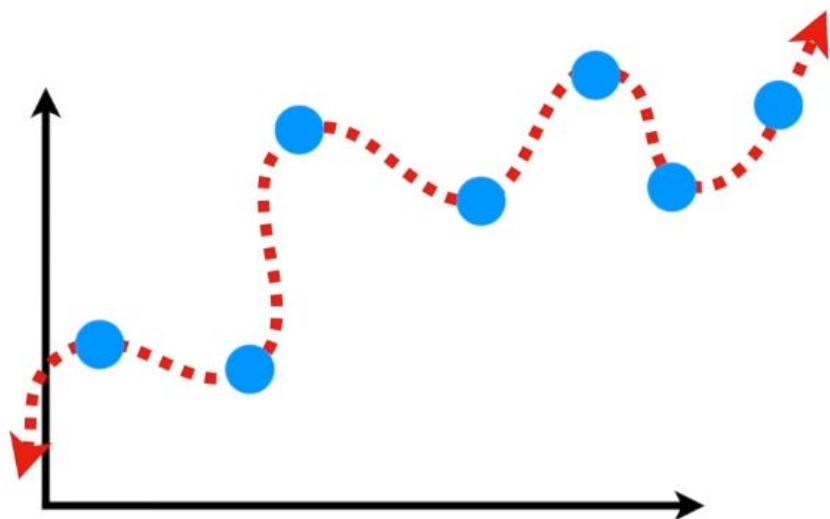
Performance on Testing Data



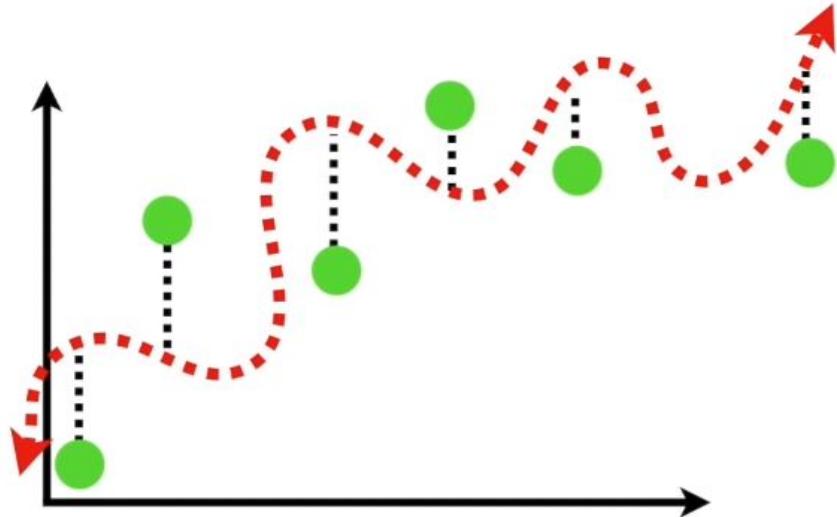
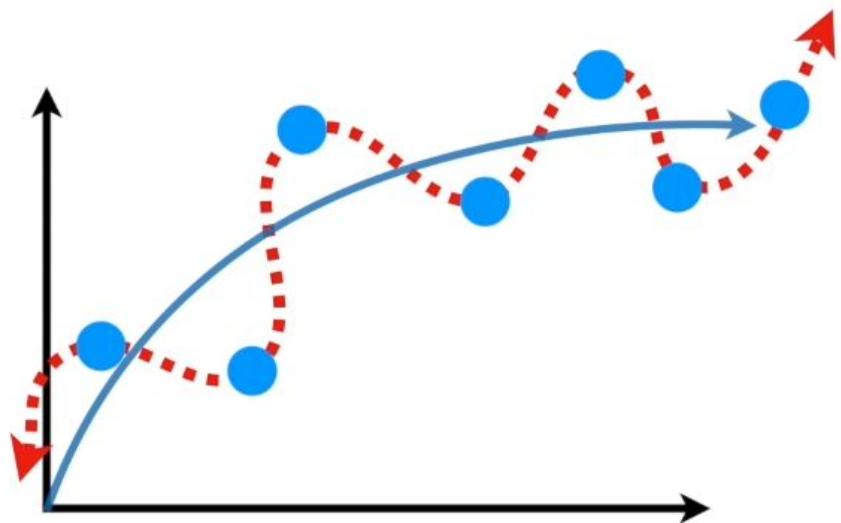
vs.



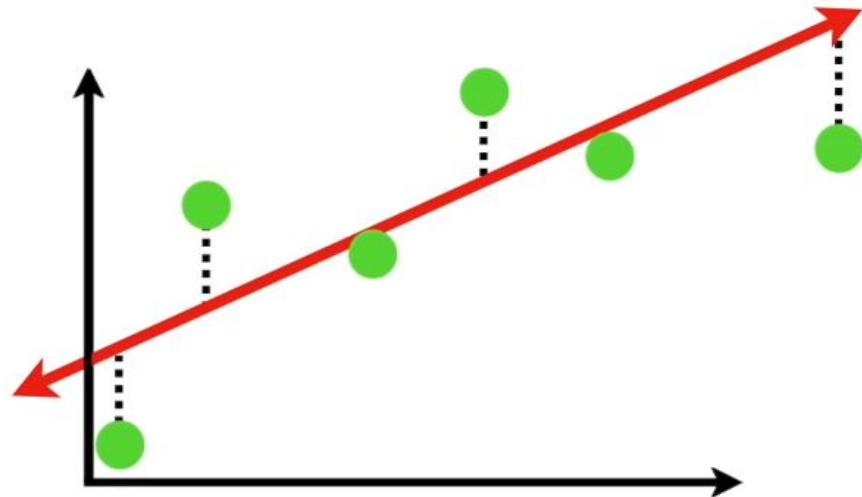
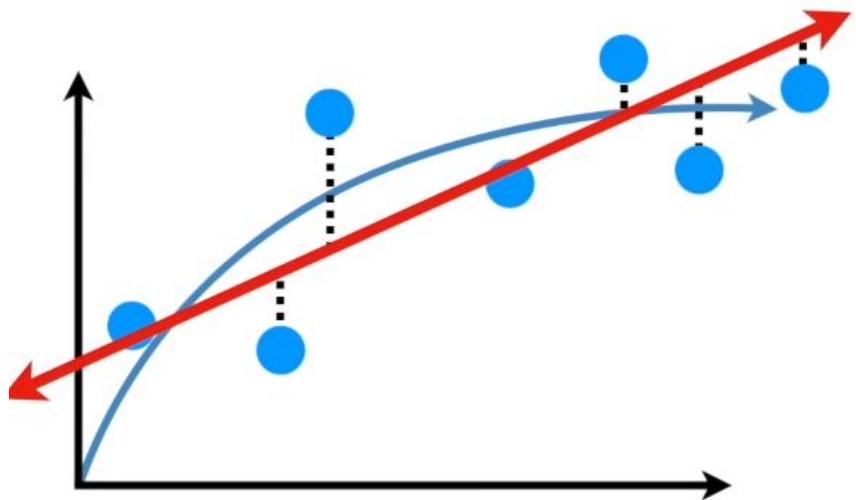
Variance



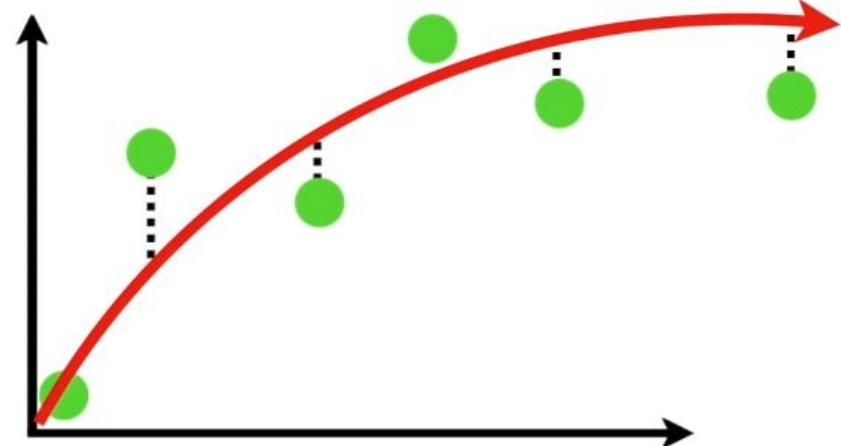
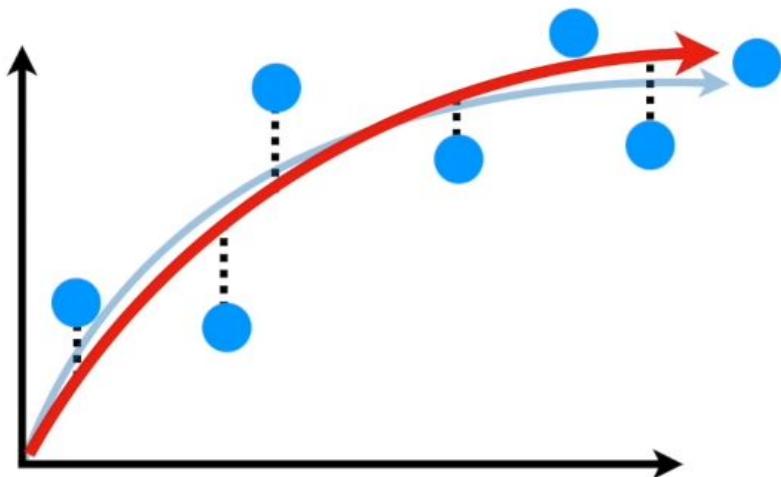
Low Bias and High Variance: Overfit



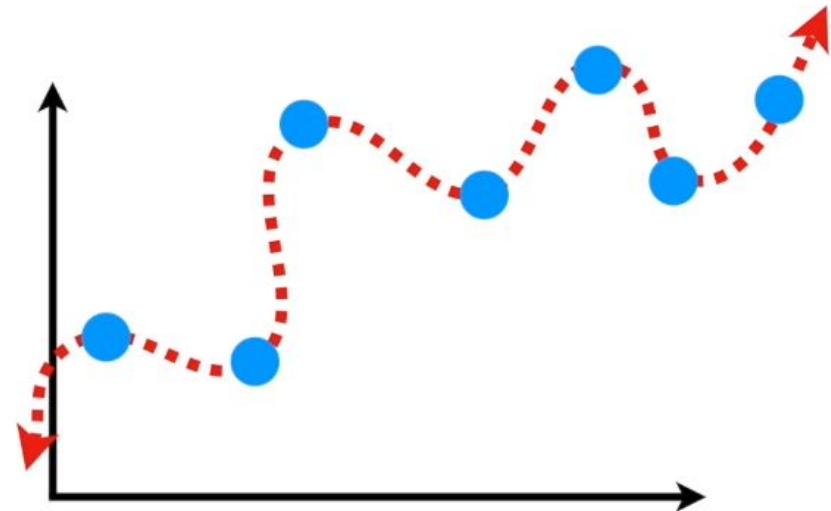
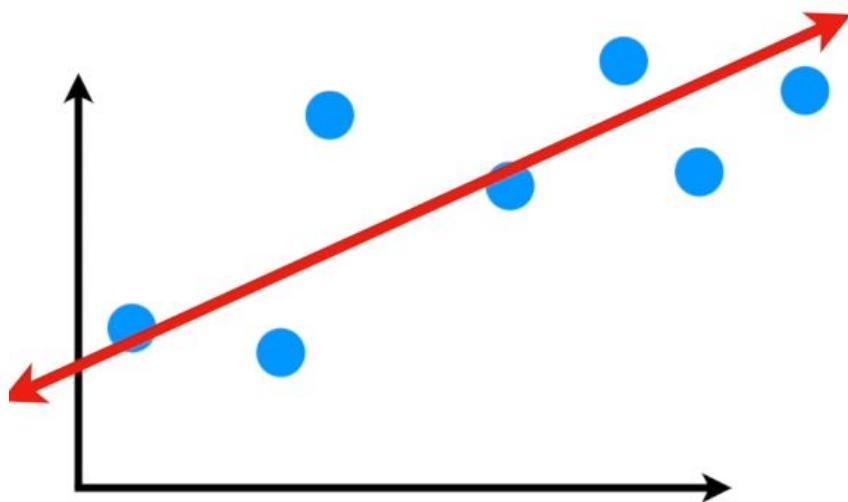
High Bias and Low Variance: Underfit



Ideal: Low Bias and Low Variance



Optimal Solution in between



Occam's Razor



Occam's Razor

The cyclic multiverse has multiple branes - each a universe - that collided, causing Big Bangs. The universes bounce back and pass through time, until they are pulled back together and again collide, destroying the old contents and creating them anew.

God did it.

Practice Question – I



Question: Suppose your model is demonstrating high variance across different training sets. Which of the following is NOT a valid way to try and reduce the variance?

- A. Increase amount of training data in each training set
- B. Change the loss function for error minimisation
- C. Decrease model complexity
- D. Reduce noise in the training dataset

Practice Question – I



Answer: B

- High variance across different sets implies that it is a case of overfitting in the model.
 - Due to this, we can do all of the options A, C and D to reduce the overfitting constraint.
- However, we cannot change the loss function used for error minimisation since it does not have a direct effect on variance encountered by the model across different training sets.

Practice Question – II



Question: We are given a small dataset. The performance on this data after training a linear model is not satisfactory. Is switching to a polynomial model a reasonable option to try?



Practice Question – II



Question: We are given a small dataset. The performance on this data after training a linear model is not satisfactory. Is switching to a polynomial model a reasonable option to try?

Answer: No, it is not a reasonable option to try.

- Polynomial model is likely to overfit on the small dataset, will not generalize.
- Might be a high variance problem. Polynomial model will increase the variance.
- Try to collect more data, if possible.
- Try to use a smaller set of features, if possible.
- We can try K-Fold cross validation.

Revisiting MSE



- Known (Observed) function is $y = f(x) + \varepsilon$, where
 - Observed value = y ; actual value = $f(x)$
 - ε is normally distributed with zero mean and standard deviation σ
- Given a set of training examples, $\{(x_i, y_i)\}$,
 - we fit an hypothesis $h(x) = w^T x + b$ to the data to minimize the squared error; $MSE = \sum_i [y_i - h(x_i)]^2$
- Given a new data point x^* (with observed value $y^* = f(x^*) + \varepsilon$), we would like to understand the expected prediction error $E[(h(x^*) - y^*)^2]$

Bias-Variance-Noise Decomposition: Lemma



- Let Z be a random variable with probability distribution $P(Z)$
- Let $\underline{Z} = E_p[Z]$ be the average value of Z
- Lemma: $E[(Z - \underline{Z})^2] = E[Z^2] - \underline{Z}^2$
- $$\begin{aligned} E[(Z - \underline{Z})^2] &= E[Z^2 - 2Z\underline{Z} + \underline{Z}^2] \\ &= E[Z^2] - 2E[Z]\underline{Z} + \underline{Z}^2 \\ &= E[Z^2] - 2\underline{Z}^2 + \underline{Z}^2 \\ &= E[Z^2] - \underline{Z}^2 \end{aligned}$$
- Corollary: $E[Z^2] = E[(Z - \underline{Z})^2] + \underline{Z}^2$



Bias-Variance-Noise Decomposition: Derivation

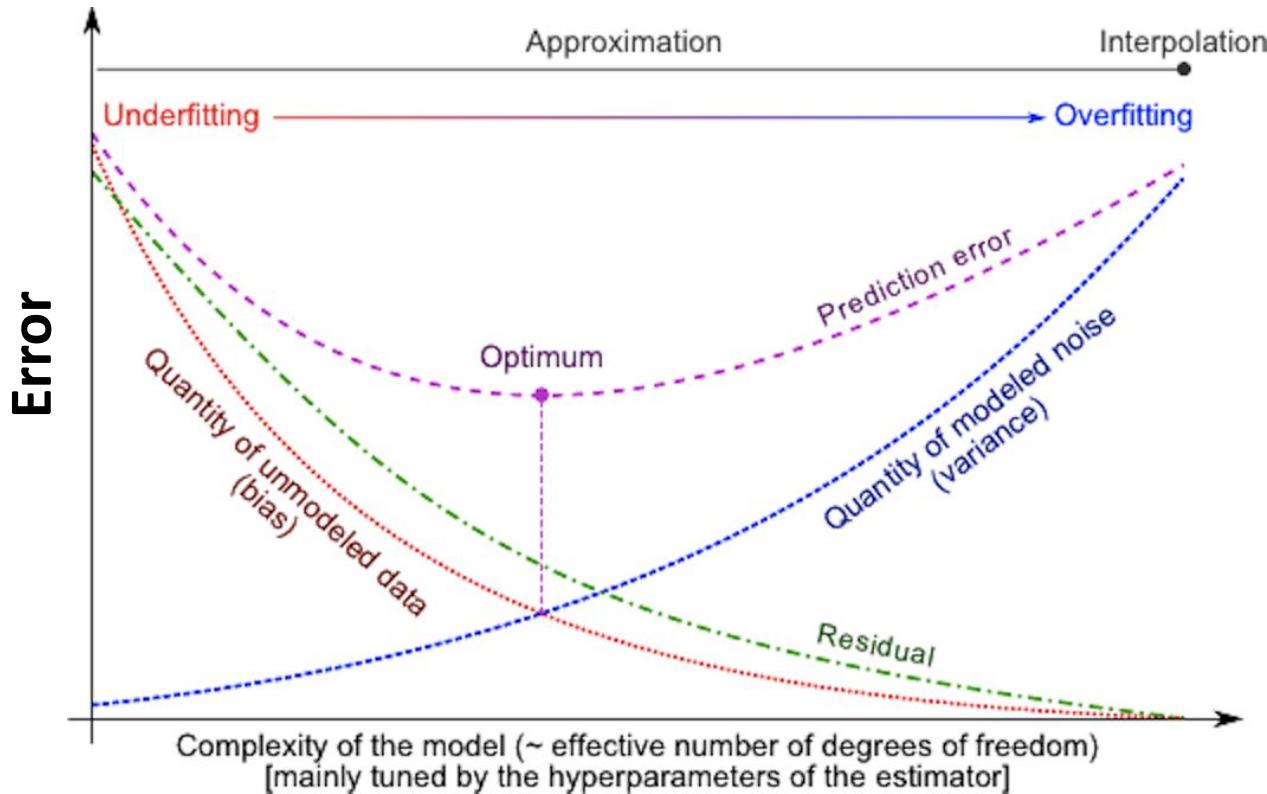
- $$\begin{aligned} E[(h(x^*) - y^*)^2] &= E[h(x^*)^2 - 2h(x^*)y^* + y^{*2}] \\ &= E[h(x^*)^2] - 2E[h(x^*)]E[y^*] + E[y^{*2}] \\ &= E[(h(x^*) - \underline{h(x^*)})^2] + \underline{h(x^*)}^2 \quad (\text{lemma}) \\ &\quad - 2\underline{h(x^*)}f(x^*) \quad (E(y^*) = E[f(x^*) + \varepsilon] = f(x^*)) \\ &\quad + E[(y^* - f(x^*))^2] + f(x^*)^2 \quad (\text{lemma}) \\ &= E[(h(x^*) - \underline{h(x^*)})^2] + [\text{variance}] \\ &\quad (\underline{h(x^*)} - f(x^*))^2 + [\text{bias}^2] \\ &E[(y^* - f(x^*))^2] [\text{noise}] \end{aligned}$$

Bias-Variance-Noise Decomposition



- $E[(h(x^*) - y^*)^2] = E[(h(x^*) - \underline{h(x^*)})^2] + (\underline{h(x^*)} - f(x^*))^2 + E[(y^* - f(x^*))^2]$, where
 $\underline{h(x^*)} = E_p [h(x^*)]$ be the average value of $h(x^*)$
 $= \text{Var}(h(x^*)) + \text{Bias}(h(x^*))^2 + E[\varepsilon^2]$
 $= \text{Var}(h(x^*)) + \text{Bias}(h(x^*))^2 + \sigma^2$
- Expected prediction error = Variance + Bias² + Noise²
- Variance = Describes how much the model varies from one training set to another training set.
- Bias² = Describes the average-error of the model.
- Noise² = Describes how much *actual value* varies from *known value*

Bias vs. Variance Analysis



Diagnostics for ML Algorithms

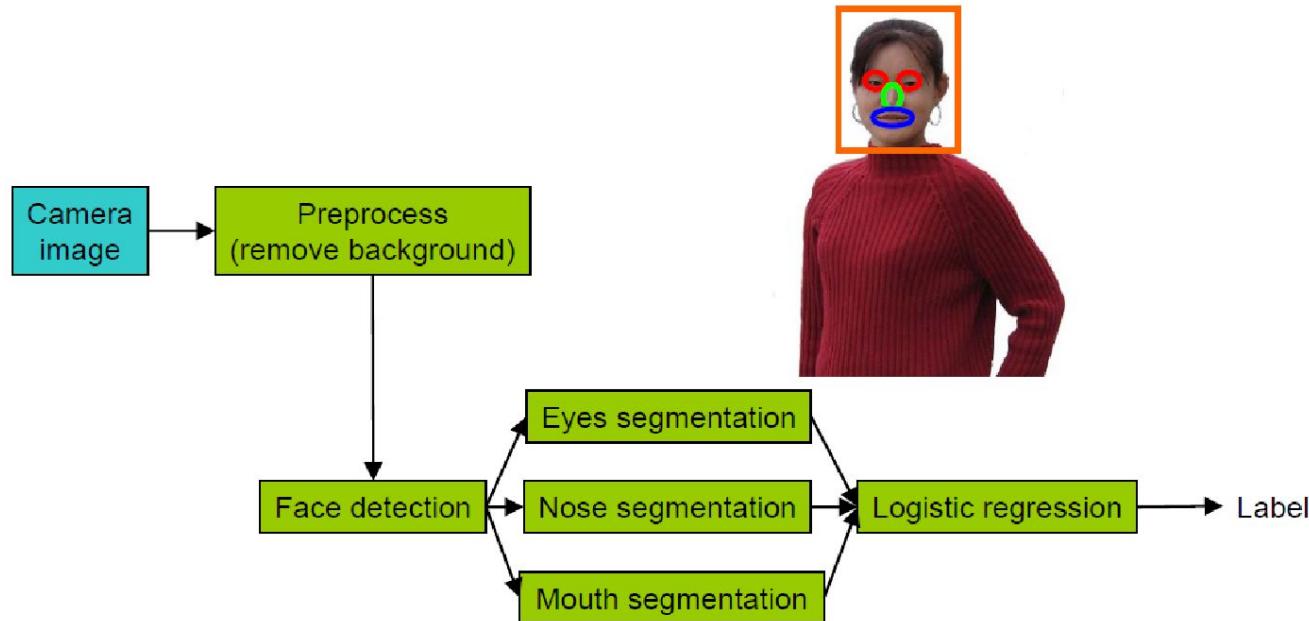


- Try getting more training examples.
 - Try a smaller set of features.
 - Try a larger set of features.
 - Try changing the features.
 - Run gradient descent for more iterations.
 - Try Newton's method instead of gradient descent.
 - Use a different value for reg. parameter λ .
 - Try using a different model (e.g., SVM).
-
- Fixes high variance.
 - Fixes high variance.
 - Fixes high bias.
 - Fixes high bias.
 - Fixes optimization algorithm.
 - Fixes optimization algorithm.
 - Fixes optimization objective.
 - Fixes optimization objective

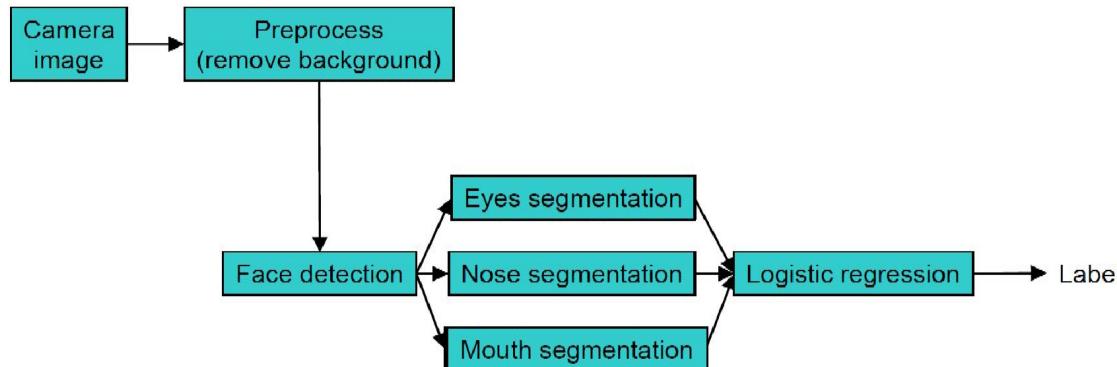
Debugging ML Systems



- Many applications combine many different learning components into a “pipeline”, e.g., Face recognition from images [toy example].



Error Analysis

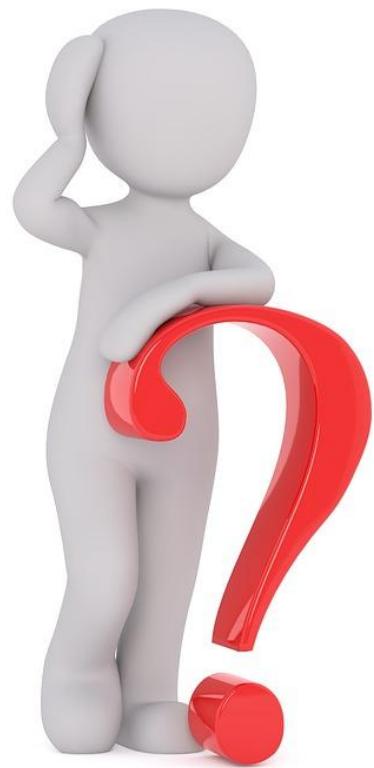


How much error is attributable to each of the components?

Plug in ground-truth for each component, and see how accuracy changes.

Conclusion: Most room for improvement in face detection and eyes segmentation.

Component	Accuracy
Overall system	85%
Preprocess (remove background)	85.1%
Face detection	91%
Eyes segmentation	95%
Nose segmentation	96%
Mouth segmentation	97%
Logistic regression	100%



Machine Learning in Practise



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



Resampling: Bootstrapping



- In practice (unlike in theory), we have only ONE training set S .
- We can simulate multiple training sets by bootstrap replicates
 - $S' = \{x \mid x \text{ is drawn at random with replacement from } S\}$ and $|S'| = |S|$

Original	Bootstrap1	Bootstrap2	Bootstrap3	Bootstrap4
1	1	2	1	1
2	1	3	2	1
3	3	3	3	1
4	3	3	5	4
5	5	4	5	5



Bootstrapping Sample: Example



- Numerical: $X = \{10, 27, 31, 40, 46\}$
- Calculate bootstrap samples
- Are these valid bootstrap samples?
 - $X_1 = \{10, 10, 31, 31, 46\}$?
 - $X_2 = \{10, 27, 31, 10\}$?
 - $X_3 = \{31, 31, 31, 31\}$?
 - $X_4 = \{31, 31, 31, 31, 31\}$?

Procedure: Measuring Bias and Variance



- Construct B bootstrap replicates of S (e.g., Construct B bootstrap replicates of S (e.g., $b = 200$): S_1, \dots, S_B)
- Apply learning algorithm to each replicate S_b to obtain hypothesis h_b
- Let $T = S \setminus S_b$ be the data points that do not appear in any $S_1..S_B$ (out of bag points).
- Compute predicted value $h_b(x)$ for each x in T
- Alternately, set T could also be a hold out set created from S before constructing bootstrap samples
- For each data point x in T , we will now have the observed corresponding value y and several predictions y_1, \dots, y_B

Procedure: Measuring Bias and Variance



- Compute the average prediction \underline{h} .
 - $\underline{h} = \sum_b h_b(x)/B$
- Estimate bias as $(\underline{h} - y)$
- Estimate variance as
 - $\sum_b (y_b - \underline{h})^2/(B - 1)$
- Assume noise is 0
- Assumptions:
 - Bootstrap replicates are not real data
 - We ignore the noise



Practice Question



Given data points: $\{76, 60, 82, 12, 38, 73, 82, 17\}$, construct **8** bootstrap samples. Report the standard error between the true mean and the average of means of bootstrapped samples.



Linear Regression Revisited



- Objective Function: An optimization problem

$$\underset{\theta}{\text{minimize}} \quad J(\theta)$$

- Minimize sum of costs over all input/output pairs

$$J(\Theta) = 1/M \sum_{i=1}^M (\Theta^T \Phi(x_i) - y_i)^2 \quad \frac{1}{M} \sum_{i=1}^M \left[\sum_{j=0}^p \theta_j x_j^{(i)} - y^{(i)} \right]$$

- Gradient Descent

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}, \text{ for all } j$$

Regularization



- To overcome underfitting:
 - Add new parameters to our model
 - Increase the model complexity
- To overcome overfitting:
 - Reduce the model complexity
 - Regularization/shrinkage
 - Change the error function to penalize hypothesis complexity

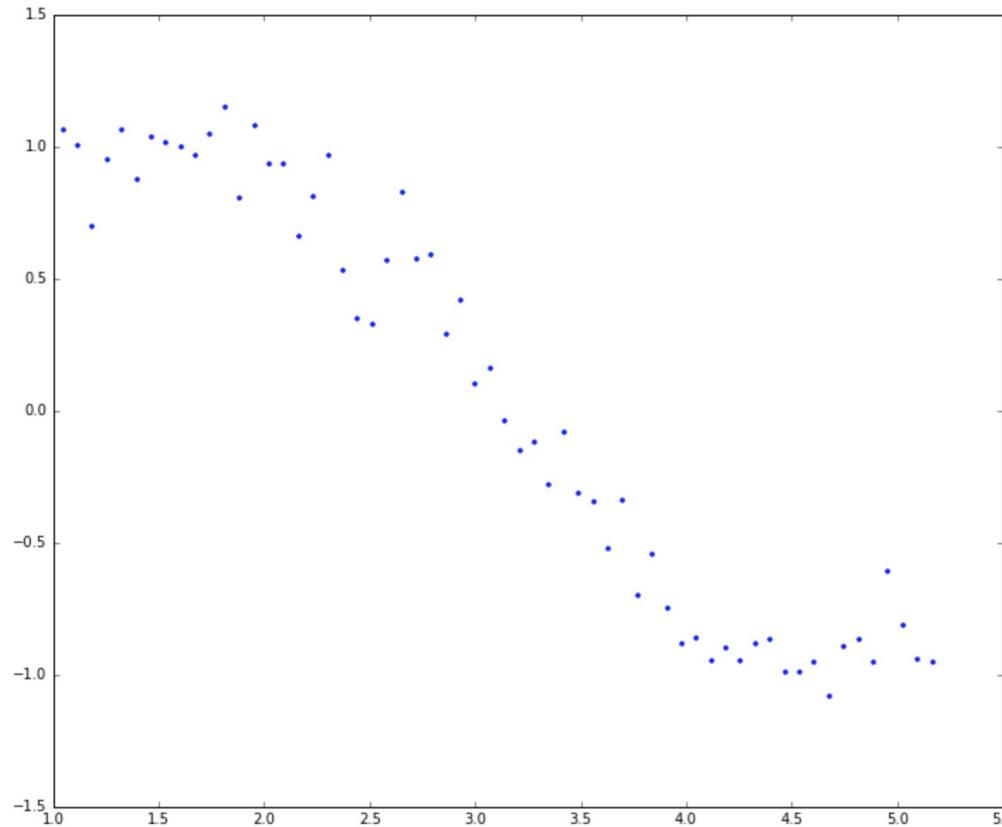
$$J(\Theta) = J_w(\Theta) + \lambda J_{pen}(\Theta)$$

Regularization

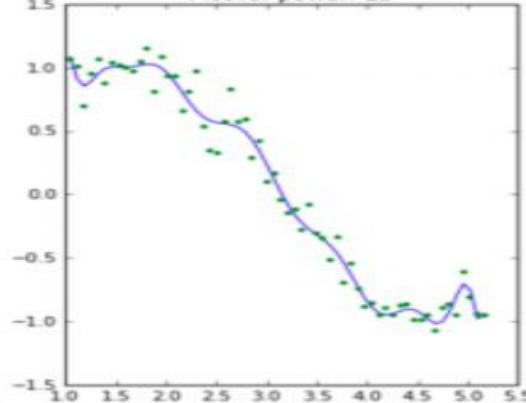
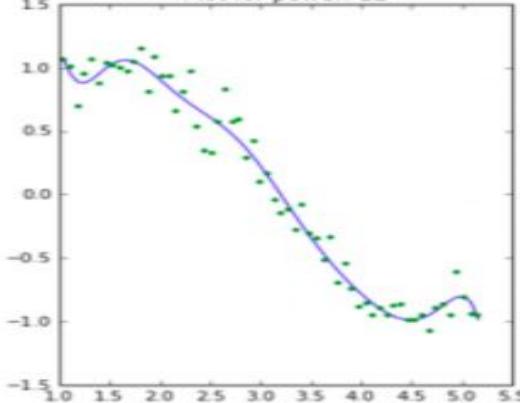
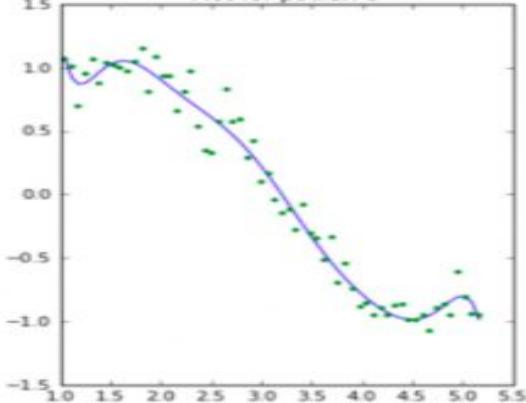
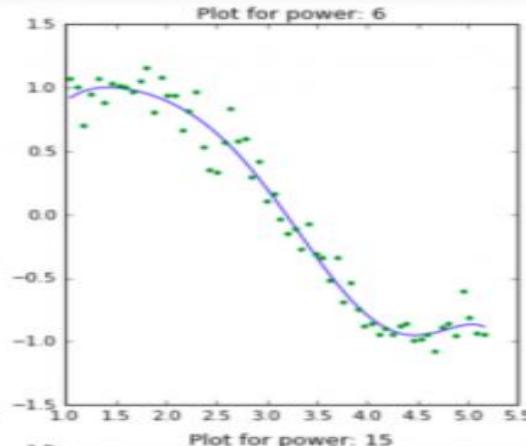
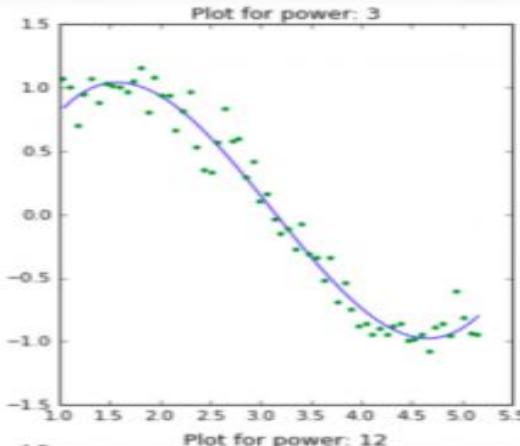
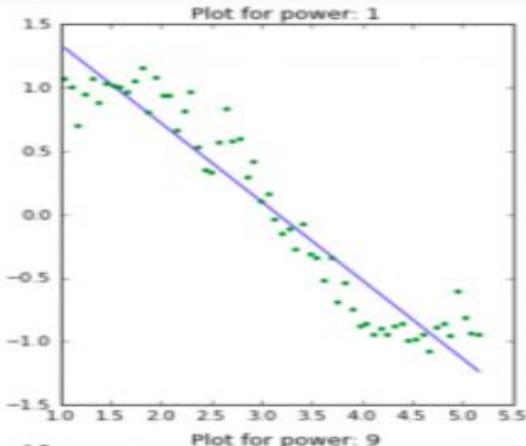


- Regularization constraints or regularizes the coefficient estimates
 - shrinks the coefficient estimates towards zero
- Shrinking the coefficient estimates can significantly reduce their variance.

SINE Curve: Noisy Data



Regression - Without any penalisation



Coefficients increase exponentially



	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	c
model_pow_1	3.3	2	-0.62	NaN	NaN	NaN								
model_pow_2	3.3	1.9	-0.58	-0.006	NaN	NaN	NaN							
model_pow_3	1.1	-1.1	3	-1.3	0.14	NaN	NaN	NaN						
model_pow_4	1.1	-0.27	1.7	-0.53	-0.036	0.014	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
model_pow_5	1	3	-5.1	4.7	-1.9	0.33	-0.021	NaN	NaN	NaN	NaN	NaN	NaN	NaN
model_pow_6	0.99	-2.8	9.5	-9.7	5.2	-1.6	0.23	-0.014	NaN	NaN	NaN	NaN	NaN	NaN
model_pow_7	0.93	19	-56	69	-45	17	-3.5	0.4	-0.019	NaN	NaN	NaN	NaN	NaN
model_pow_8	0.92	43	-1.4e+02	1.8e+02	-1.3e+02	58	-15	2.4	-0.21	0.0077	NaN	NaN	NaN	NaN
model_pow_9	0.87	1.7e+02	-6.1e+02	9.6e+02	-8.5e+02	4.6e+02	-1.6e+02	37	-5.2	0.42	-0.015	NaN	NaN	NaN
model_pow_10	0.87	1.4e+02	-4.9e+02	7.3e+02	-6e+02	2.9e+02	-87	15	-0.81	-0.14	0.026	-0.0013	NaN	NaN
model_pow_11	0.87	-75	5.1e+02	-1.3e+03	1.9e+03	-1.6e+03	9.1e+02	-3.5e+02	91	-16	1.8	-0.12	0.0034	NaN
model_pow_12	0.87	-3.4e+02	1.9e+03	-4.4e+03	6e+03	-5.2e+03	3.1e+03	-1.3e+03	3.8e+02	-80	12	-1.1	0.062	-1
model_pow_13	0.86	3.2e+03	-1.8e+04	4.5e+04	-6.7e+04	6.6e+04	-4.6e+04	2.3e+04	-8.5e+03	2.3e+03	-4.5e+02	62	-5.7	0
model_pow_14	0.79	2.4e+04	-1.4e+05	3.8e+05	-6.1e+05	6.6e+05	-5e+05	2.8e+05	-1.2e+05	3.7e+04	-8.5e+03	1.5e+03	-1.8e+02	1
model_pow_15	0.7	-3.6e+04	2.4e+05	-7.5e+05	1.4e+06	-1.7e+06	1.5e+06	-1e+06	5e+05	-1.9e+05	5.4e+04	-1.2e+04	1.9e+03	-1

Ridge Regularization



- Minimization objective = LS Obj + $\lambda * (\text{sum of square of slope})$

$$J(\Theta) = \frac{1}{M} \sum_{i=1}^M (\Theta^T \Phi(x_i) - y_i)^2 + \lambda \sum_{j=1}^p \Theta_j^2$$

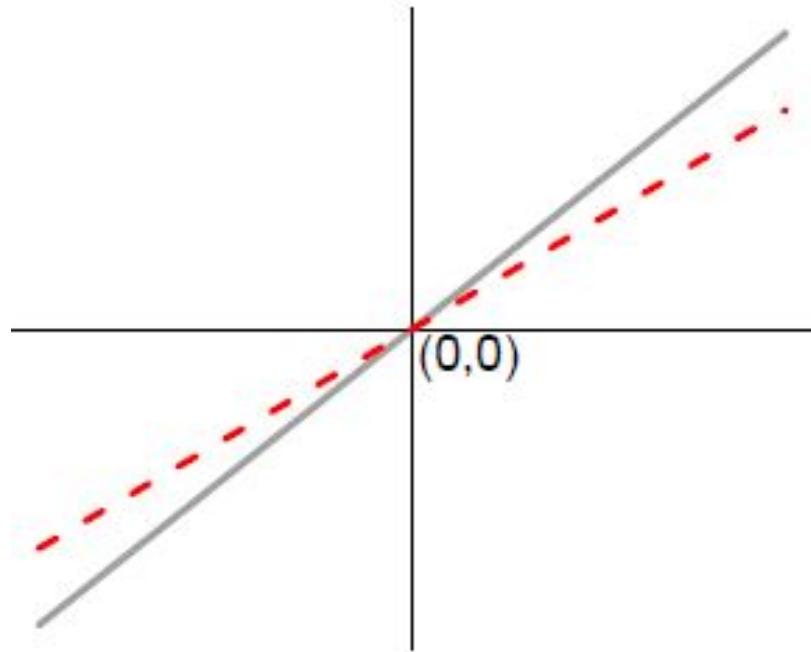
$$\frac{\partial J}{\partial \Theta} = \frac{2}{M} \sum_{i=1}^M (\Theta^T \Phi(x_i) - y_i) \Phi(x_i) + 2\lambda \Theta_j$$

$$\Theta(j+1) = (1 - 2\lambda\alpha)\Theta_j - \frac{2\alpha}{M} \sum_{i=1}^M (\Theta^T \Phi(x_i) - y_i) \Phi(x_i)$$

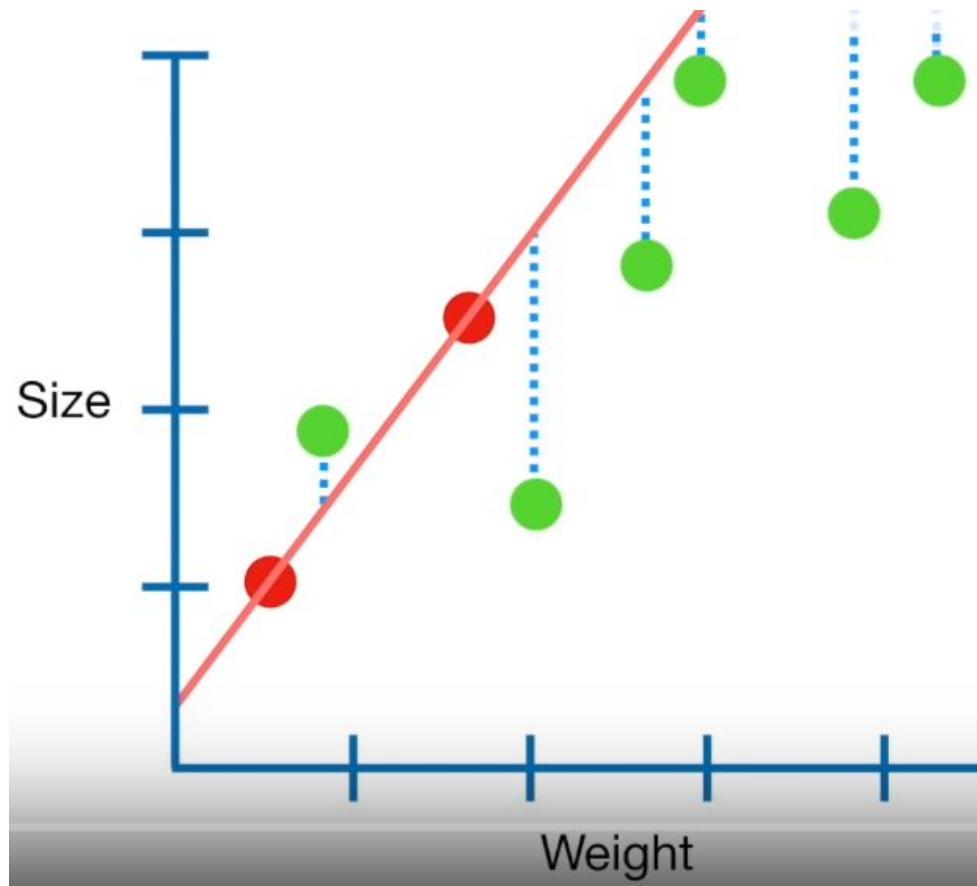
Ridge Regularization



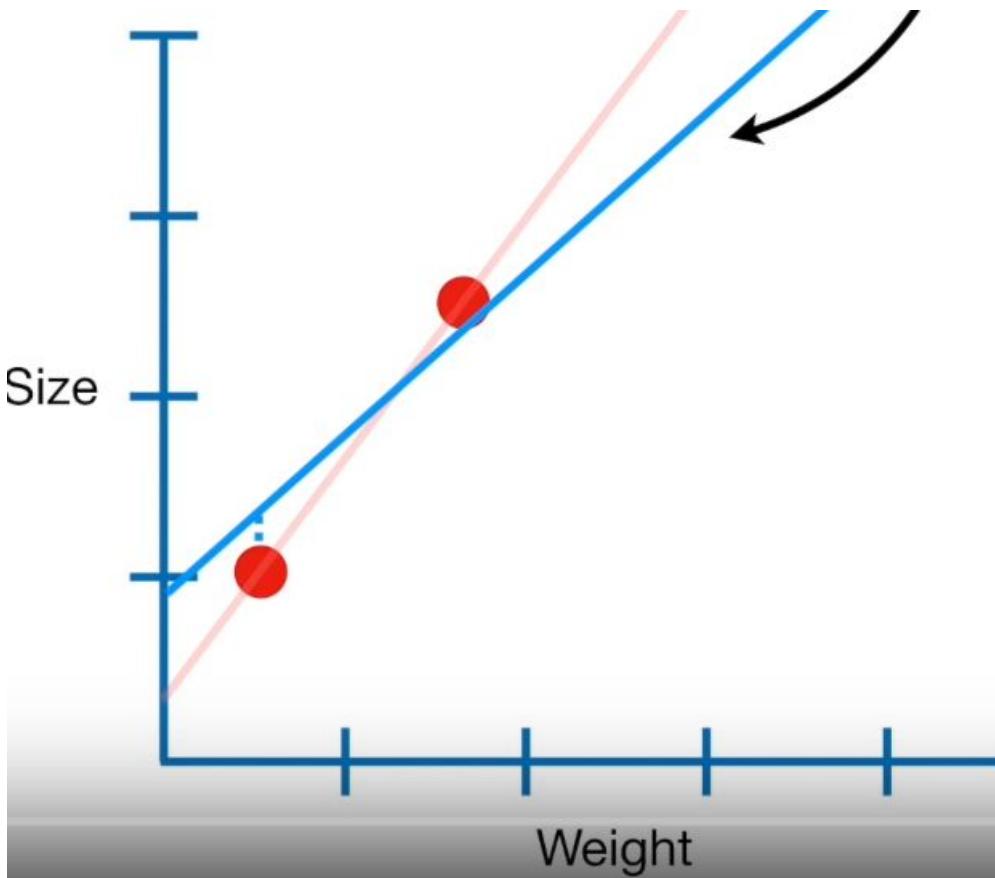
Ridge



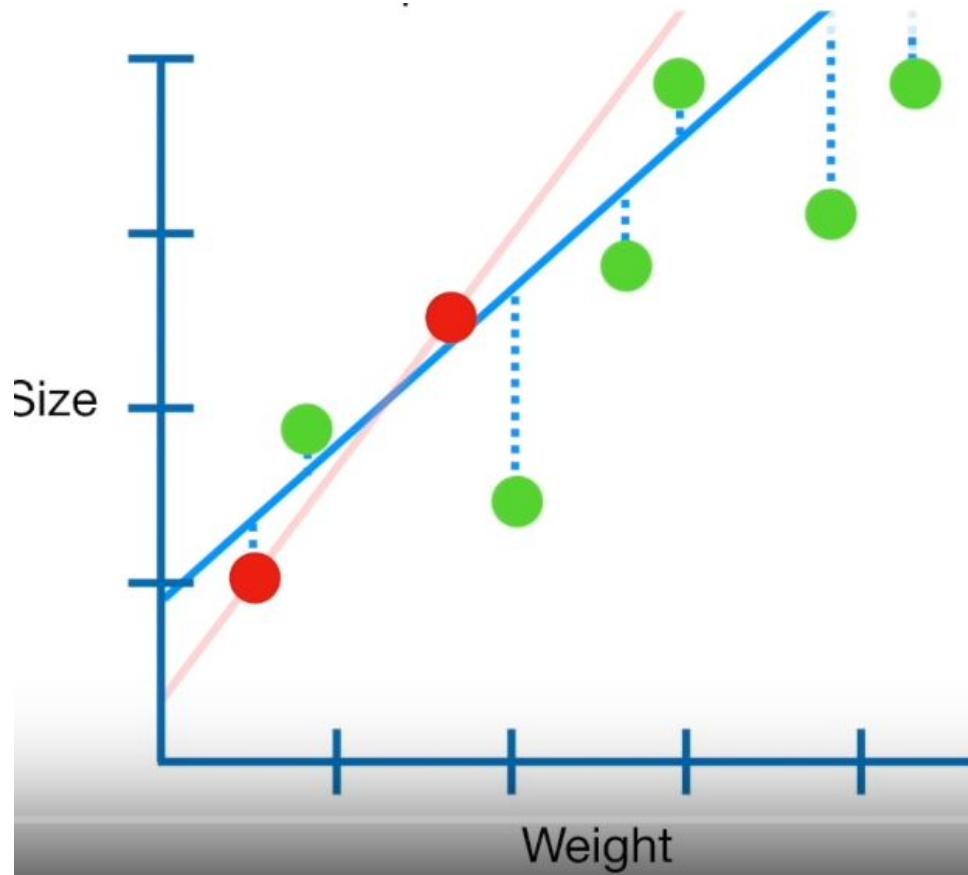
Low Bias and High Variance: Overfit



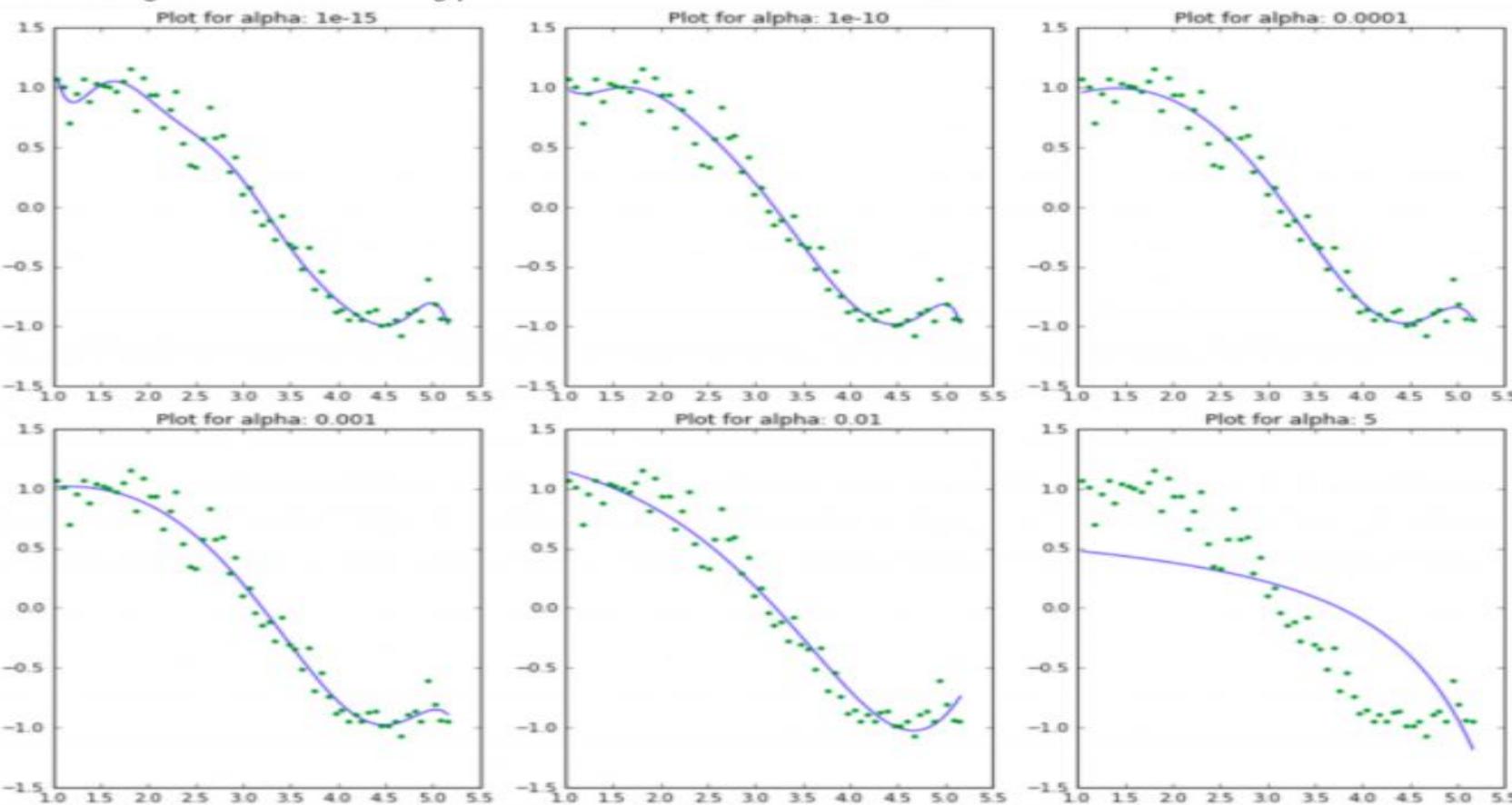
Ridge Regression



Ridge Regression



Regression - Ridge Regression



Regression - Ridge Regression



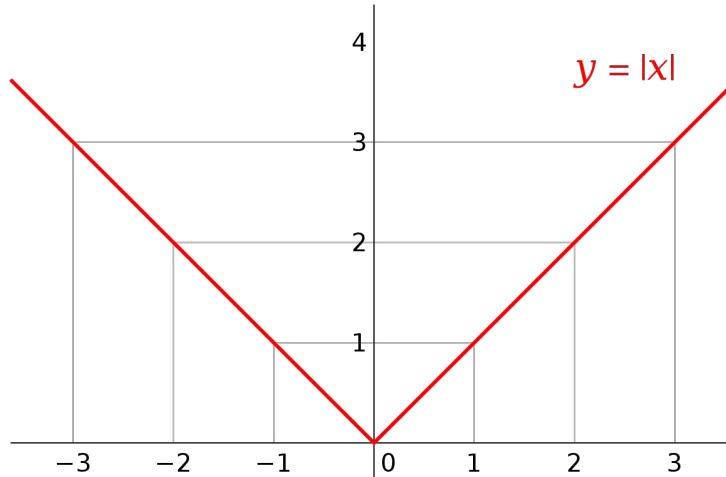
	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	coef_x_12	coef_x_13	coef_x_14	coef_x_15	coef_x_16	coef_x_17	coef_x_18	coef_x_19	coef_x_20
alpha_1e-15	0.87	95	-3e+02	3.8e+02	-2.4e+02	66	0.96	-4.8	0.64	0.15	-0.026	-0.0054	0.00086	0.0	0.00086	0.0	0.00086	0.0	0.00086	0.0	0.00086	0.0
alpha_1e-10	0.92	11	-29	31	-15	2.9	0.17	-0.091	-0.011	0.002	0.00064	2.4e-05	-2e-05	-4.2e-05	0.00064	2.4e-05	-2e-05	-4.2e-05	0.00064	2.4e-05	-2e-05	-4.2e-05
alpha_1e-08	0.95	1.3	-1.5	1.7	-0.68	0.039	0.016	0.00016	-0.00036	-5.4e-05	-2.9e-07	1.1e-06	1.9e-07	2e-07	0.00016	-0.00036	-5.4e-05	-2.9e-07	1.1e-06	1.9e-07	2e-07	0.00016
alpha_0.0001	0.96	0.56	0.55	-0.13	-0.026	-0.0028	-0.00011	4.1e-05	1.5e-05	3.7e-06	7.4e-07	1.3e-07	1.9e-08	1.9	7.4e-07	1.3e-07	1.9e-08	1.9	7.4e-07	1.3e-07	1.9e-08	1.9
alpha_0.001	1	0.82	0.31	-0.087	-0.02	-0.0028	-0.00022	1.8e-05	1.2e-05	3.4e-06	7.3e-07	1.3e-07	1.9e-08	1.7	7.3e-07	1.3e-07	1.9e-08	1.7	7.3e-07	1.3e-07	1.9e-08	1.7
alpha_0.01	1.4	1.3	-0.088	-0.052	-0.01	-0.0014	-0.00013	7.2e-07	4.1e-06	1.3e-06	3e-07	5.6e-08	9e-09	1.1	7.2e-07	4.1e-06	1.3e-06	3e-07	5.6e-08	9e-09	1.1	7.2e-07
alpha_0.1	5.6	0.97	-0.14	-0.019	-0.003	-0.00047	-7e-05	-9.9e-06	-1.3e-06	-1.4e-07	-9.3e-09	1.3e-09	7.8e-10	2.4	-9.3e-09	1.3e-09	7.8e-10	2.4	-9.3e-09	1.3e-09	7.8e-10	2.4
alpha_0.5	14	0.55	-0.059	-0.0085	-0.0014	-0.00024	-4.1e-05	-6.9e-06	-1.1e-06	-1.9e-07	-3.1e-08	-5.1e-09	-8.2e-10	-1.1	-3.1e-08	-5.1e-09	-8.2e-10	-1.1	-3.1e-08	-5.1e-09	-8.2e-10	-1.1
alpha_10	18	0.4	-0.037	-0.0055	-0.00095	-0.00017	-3e-05	-5.2e-06	-9.2e-07	-1.6e-07	-2.9e-08	-5.1e-09	-9.1e-10	-1.6	-2.9e-08	-5.1e-09	-9.1e-10	-1.6	-2.9e-08	-5.1e-09	-9.1e-10	-1.6
alpha_20	23	0.28	-0.022	-0.0034	-0.0006	-0.00011	-2e-05	-3.6e-06	-6.6e-07	-1.2e-07	-2.2e-08	-4e-09	-7.5e-10	-1.4	-2.2e-08	-4e-09	-7.5e-10	-1.4	-2.2e-08	-4e-09	-7.5e-10	-1.4

LASSO Regression



- Minimization objective = LS Obj + λ^* (sum of absolute value of slope)

$$J(\Theta) = \frac{1}{M} \sum_{i=1}^M (\Theta^T \Phi(x_i) - y_i)^2 + \lambda \sum_{j=1}^p |\Theta_j|$$



LASSO Regression



- Lasso coordinate descent - closed form solution

$$\begin{cases} \theta_{(j+1)} = \rho_j + \lambda & \text{for } \rho_j < -\lambda \\ \theta_{(j+1)} = 0 & \text{for } -\lambda \leq \rho_j \leq \lambda \\ \theta_{(j+1)} = \rho_j - \lambda & \text{for } \rho_j > \lambda \end{cases}$$

$$\rho_j = \sum_{i=1}^m x_j^{(i)} (y^{(i)} - \sum_{k \neq j}^p \theta_k x_k^{(i)})$$

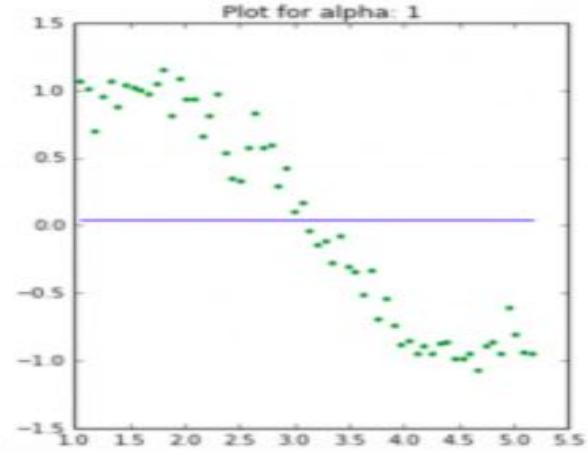
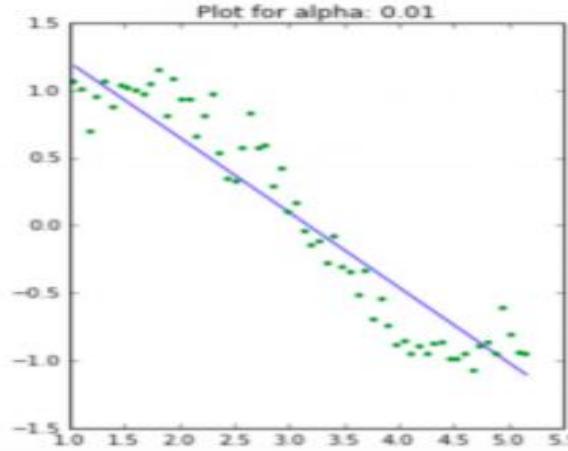
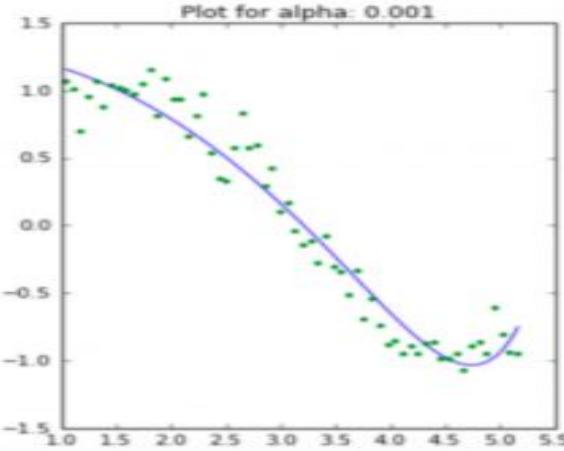
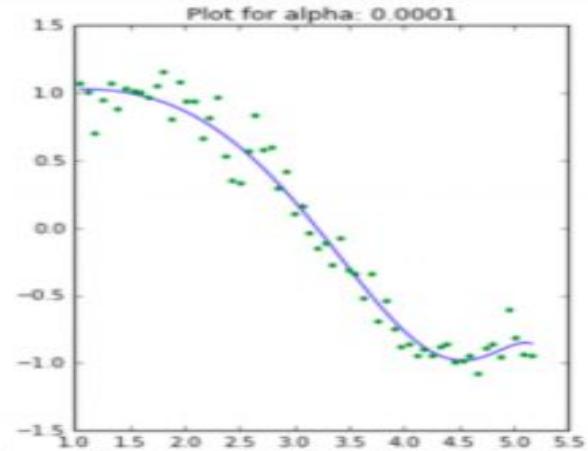
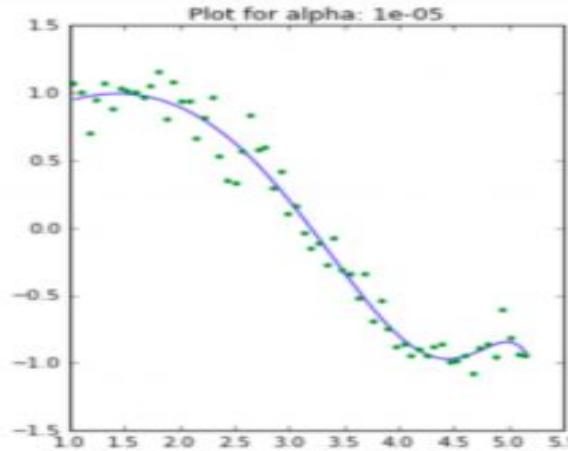
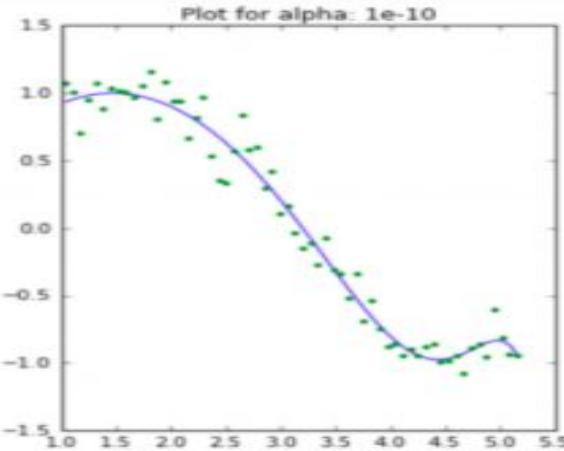


Lasso Regularization



- One significant problem of ridge regression is that the penalty term will never force any of the coefficients to be exactly zero.
- Thus, the final model will include all predictors (features), which creates a challenge in model interpretation
- The lasso works in a similar way to ridge regression, except it uses a different penalty term that shrinks some of the coefficients exactly to zero.
- Lasso performs a variable feature selection.
- $Y = m_1 * x_1 + m_2 * x_2 + m_3 * x_3 + c$

Regression - Lasso Regression



Regression - Lasso Regression



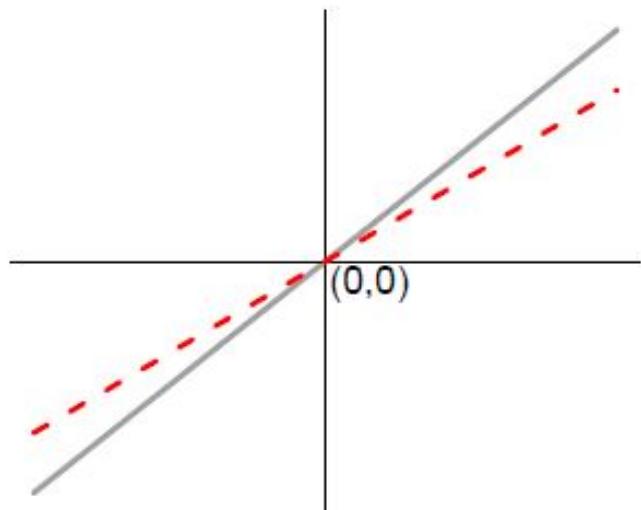
	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	co
alpha_1e-15	0.96	0.22	1.1	-0.37	0.00089	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4
alpha_1e-10	0.96	0.22	1.1	-0.37	0.00088	0.0016	-0.00012	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.4
alpha_1e-08	0.96	0.22	1.1	-0.37	0.00077	0.0016	-0.00011	-6.4e-05	-6.3e-06	1.4e-06	7.8e-07	2.1e-07	4e-08	5.3
alpha_1e-05	0.96	0.5	0.6	-0.13	-0.038	-0	0	0	0	7.7e-06	1e-06	7.7e-08	0	0
alpha_0.0001	1	0.9	0.17	-0	-0.048	-0	-0	0	0	9.5e-06	5.1e-07	0	0	0
alpha_0.001	1.7	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0	1.5e-08	7.5
alpha_0.01	3.6	1.8	-0.55	-0.00056	-0	-0	-0	-0	-0	-0	-0	0	0	0
alpha_1	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
alpha_5	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0
alpha_10	37	0.038	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0	-0

HIGH SPARSITY

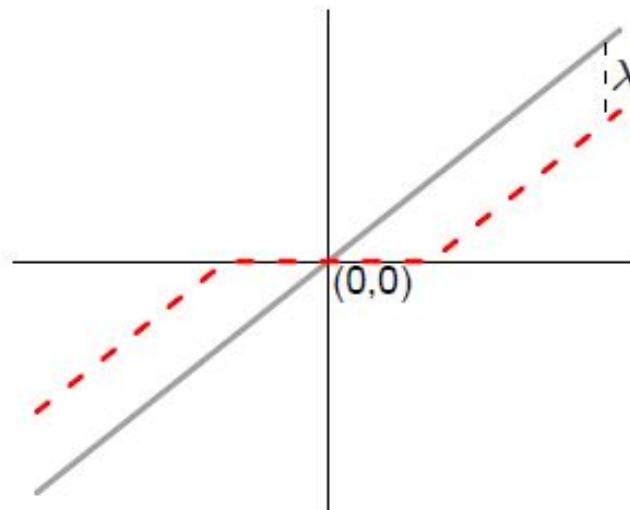
RIDGE vs LASSO Regression



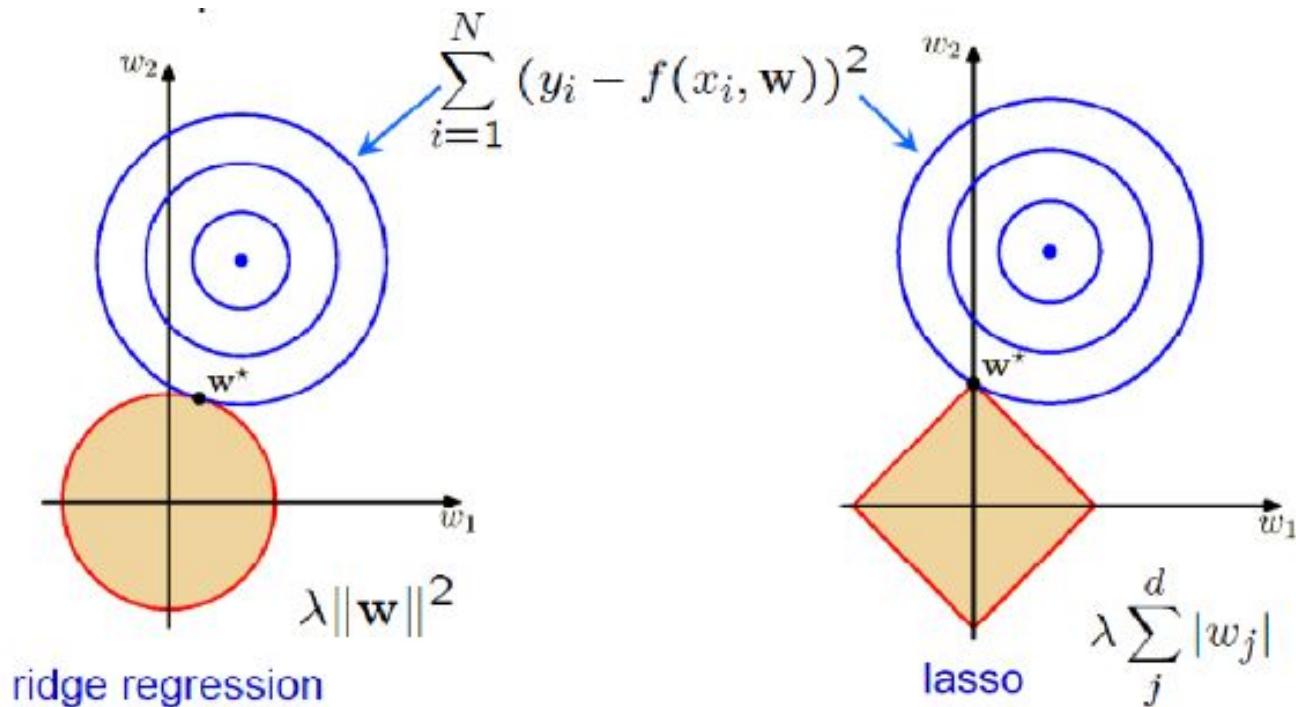
Ridge



Lasso



RIDGE vs LASSO Regression



Lasso vs Ridge Regression



- The lasso produces simpler and more interpretable models that involved only a subset of predictors.
- The lasso leads to qualitatively similar behavior to ridge regression, in that as λ increases, the variance decreases and the bias increases.
- The lasso can generate more accurate predictions compared to ridge regression.
- Cross-validation can be used in order to determine which approach is better on a particular data set.

Tuning Parameter



- Increased λ leads to increased bias but decreased variance
- $\lambda = 0$
 - The objective becomes same as simple linear regression.
- $0 < \lambda < \infty$:
 - The coefficients will be somewhere between 0 and ones for simple linear regression.

Cross-validation is used to find the parameter that results in the lowest variance.



Naïve Bayes



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



Problem Setting



- Dataset is a set of possible instances $\mathbf{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- $x_i \in \mathbf{X}$: Each sample is a vector with \mathbf{R}^d drawn from distribution $P(x, y)$
- Unknown target function $f : \mathbf{X} \rightarrow \mathbf{Y}$ as distribution $P(y/x)$
- Set of function hypotheses $H = \{ h \mid h : X \rightarrow Y \}$
- **Input:** Training examples $\{\langle x_i, y_i \rangle\}$
- **Output:** Hypothesis $h \in H$ that best approximates target function f
- Knowing the P , we can simply bayes theorem to get a perfect hypothesis.
 - $P_\theta(x, y) \sim P(x, y)$
 - We do not have $P(x, y)$ but we do have \mathbf{D}

How to get θ ?



- **Maximum Likelihood Estimation (MLE)** gives us the solution which maximises the likelihood.
 - Find θ that maximizes the probability of the data D
 - $\text{argmax}_{\theta} P_{\theta}(D)$
- **Maximum A Posterior (MAP)** gives us the solution which maximises the posterior probability.
 - Find θ that is most likely given the data D .
 - $P(\theta|D) = P(D|\theta) * P(\theta)/P(D)$
 - Assumes the availability of the prior $P(\theta) \sim N(\mu, \sigma^2)$
- Both ML and MAP return only single and specific values for the parameter θ !

How to get θ ?



- **Bayesian Inference**

- We are not estimating in estimating θ s, but in making predictions!

$$p_{\theta}(y|X = x)$$

- Holy grail of all predictions will average out all possible models one could think of!

$$p(y|X = x) = \int_{\theta} p(y|\theta)p(\theta|D)d\theta$$



Introduction



- Rev. Thomas Bayes (1702–61)
 - Existence of God
- Why ‘Naïve’?
 - Features are independent of each other
- Pros:
 - Easy and fast, performs well in multi class prediction
 - Better to other models like logistic regression and need less training data.



Naïve Bayes Model



- Consider each attribute and class label as random variables
- Given a record with attributes (X_1, X_2, \dots, X_n)
 - Goal is to predict class Y
 - Specifically, we want to find the value of Y that maximizes
 - $P(Y = y_1 | X_1, X_2, \dots, X_n)$
 - $P(Y = y_2 | X_1, X_2, \dots, X_n)$
 - $P(Y = y_3 | X_1, X_2, \dots, X_n)$
- Can we estimate $P(Y = y_i | X_1, X_2, \dots, X_n)$ directly from data?

Naïve Bayes Model



- Compute the posterior probability $P(Y| X_1, X_2, \dots X_n)$ for all values of Y using the Bayes theorem
 - $P(Y| X_1, X_2, \dots X_n) =$
 - $P(X_1, X_2, \dots X_n | Y) * P(Y) / P(X_1, X_2, \dots X_n)$
- Choose value of Y that maximizes $P(Y| X_1, X_2, \dots X_n)$
 - Equivalent to choosing value of Y that maximizes $P(X_1, X_2, \dots X_n | Y) * P(Y)$
- How to estimate $P(X_1, X_2, \dots X_n | Y) * P(Y)$?

Naïve Bayes Model



- Assume independence among attributes X_i when class is given $P(X_1, X_2, \dots, X_n | Y_j) * P(Y = Y_j) =$
 - $P(Y = Y_j) * P(X_1 | Y_j) * P(X_2 | Y_j) * \dots * P(X_n | Y_j)$
- Can estimate $P(X_i | Y_j)$ for all X_i and Y_j .
- New point is classified to Y_j if $P(Y_j) \prod P(X_i | Y_j)$ is maximum.

Naïve Bayes Classifier



$$Y^{new} = \operatorname{argmax} P(Y = Y_j) \prod_i P(X^{new}|Y = Y_j)$$



Parameter Estimation: Discrete (MLE)



- Training in Naïve Bayes is **easy**:
 - Estimate $P(Y=y_j)$ as the fraction of records with $Y=v$

$$P(Y = y_j) = \frac{\text{count}(Y=y_j)}{n}$$

- Estimate $P(X_i=x_{ij}|Y=y_j)$ as the fraction of records with $Y=y_j$ for which $X_i=x_{ij}$

$$P(X = x_{ij}|Y = y_j) = \frac{\text{count}(X=x_{ij} \wedge Y=y_j)}{\text{count}(X=x_{ij})}$$

Example: $P(\text{Red}|\text{Yes})$



Colour	Type	Origin	Stolen
Red	Sports	Domestic	Yes
Red	Sports	Domestic	No
Red	Sports	Domestic	Yes
Yellow	Sports	Domestic	No
Yellow	Sports	Imported	Yes
Yellow	SUV	Imported	No
Yellow	SUV	Imported	Yes
Yellow	SUV	Domestic	No
Red	SUV	Imported	No
Red	Sports	Imported	Yes

Parameter Estimation: MAP/Smoothing



- If one of the conditional probability is zero, then the entire expression becomes zero
 - c : *number of classes*
 - p : *prior probability*
 - m : *parameter*

$$\text{Laplace : } P(A_i | C) = \frac{N_{ic} + 1}{N_c + c}$$

$$\text{m - estimate : } P(A_i | C) = \frac{N_{ic} + mp}{N_c + m}$$

Parameter Estimation: Continuous



- Gaussian naive Bayes

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

- Bernoulli naive Bayes

$$p(\mathbf{x} \mid C_k) = \prod_{i=1}^n p_{ki}^{x_i} (1 - p_{ki})^{(1-x_i)}$$

- Multinomial naive Bayes

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}^{x_i}$$



Parameter Estimation: Numerical



Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2}$$

$P(Income = 120 | No)$

Parameter Estimation: Numerical



$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

$$= \frac{1}{\sqrt{2\pi}(54.54)} e^{-\frac{(120-110)^2}{2(2975)}} = 0.0072$$

Example: (Red, Domestic, SUV)?



Colour	Type	Origin	Stolen
Red	Sports	Domestic	Yes
Red	Sports	Domestic	No
Red	Sports	Domestic	Yes
Yellow	Sports	Domestic	No
Yellow	Sports	Imported	Yes
Yellow	SUV	Imported	No
Yellow	SUV	Imported	Yes
Yellow	SUV	Domestic	No
Red	SUV	Imported	No
Red	Sports	Imported	Yes

Example: (Red, Domestic, SUV)?



- $P(\text{Yes}) = 0.5$ and $P(\text{No}) = 0.5$
- $p = 1 / (\text{number-of-attribute-values}) = 0.5$ for all of our attributes
- $m = 3$
- $P(\text{Red}|\text{Yes})$, $P(\text{SUV}|\text{Yes})$, $P(\text{Domestic}|\text{Yes})$,
- $P(\text{Red}|\text{No})$, $P(\text{SUV}|\text{No})$, and $P(\text{Domestic}|\text{No})$

$$P(\text{Red}|\text{Yes}) = \frac{3 + 3 * .5}{5 + 3} = .56$$

$$P(\text{Red}|\text{No}) = \frac{2 + 3 * .5}{5 + 3} = .43$$

$$P(\text{SUV}|\text{Yes}) = \frac{1 + 3 * .5}{5 + 3} = .31$$

$$P(\text{SUV}|\text{No}) = \frac{3 + 3 * .5}{5 + 3} = .56$$

$$P(\text{Domestic}|\text{Yes}) = \frac{2 + 3 * .5}{5 + 3} = .43$$

$$P(\text{Domestic}|\text{No}) = \frac{3 + 3 * .5}{5 + 3} = .56$$

Summary



- Robust to isolated noise points
- Handle missing values by ignoring the instance during probability estimate calculations
- Robust to irrelevant attributes
- Independence assumption may not hold for some attributes
 - Use other techniques such as Bayesian Belief Networks (BBN)



Decision Trees



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



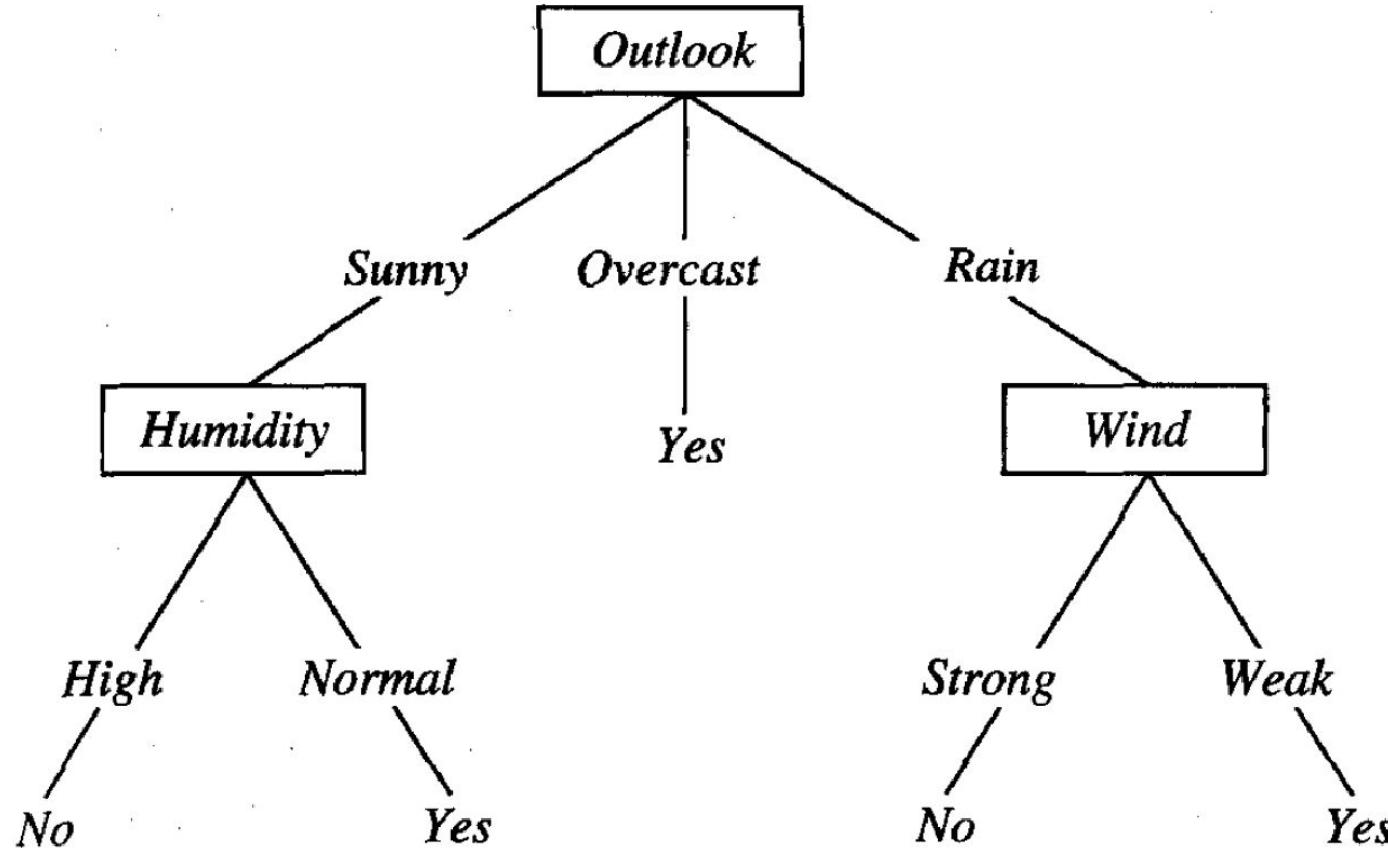
Playing Tennis



PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Playing Tennis





Why DT?

- Interpretable model
 - Simple and visual
- Lower computational complexity
- Exploratory analysis
 - Accuracy is not a concern
- Data is non-parametric in nature
 - Does not require any assumptions on the distribution of data



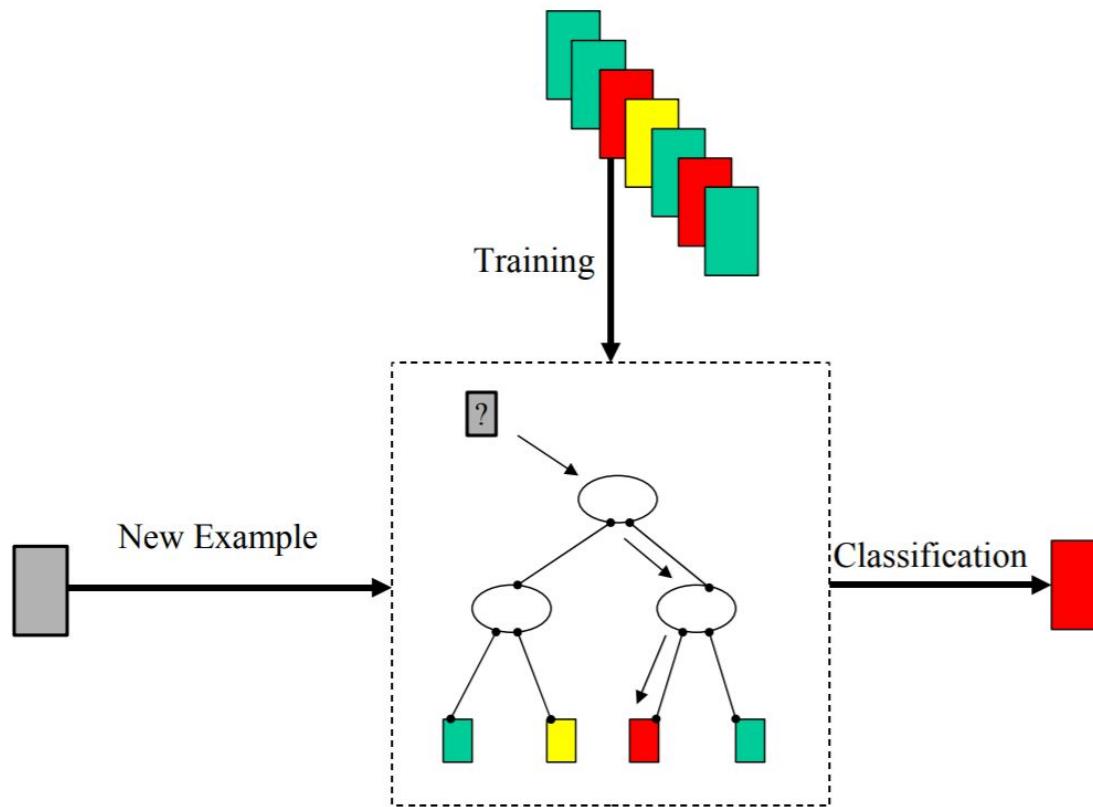
Examples



- Approve a credit card application or not
- Approve a loan application or not
- If a customer will take up a product or not
- If a transaction is fraudulent or not
- If a customer will close a mobile/telephone connection



Decision Tree Learning



Decision Tree Learning



A Decision tree for

$F: \langle \text{Outlook, Humidity, Wind, Temp} \rangle$ PlayTennis?

- Each internal node: test one attribute X_i
 - $\langle \text{Outlook, Humidity, Wind, Temp} \rangle$
- Each branch from a node: selects one value for X_i
 - $\langle \text{Outlook: Sunny, Overcast, Rain} \rangle$
- Each leaf node: predict Y (or $P(Y|X \in \text{leaf})$)
 - Given $\langle \text{Outlook: Rainy Wind:Week} \rangle$
 - PlayTennis = Yes

Problem Setting



- Set of possible instances X
 - each instance x in X is a feature vector
 - e.g., <Humidity=low, Wind=weak, Outlook=rain, Temp=hot>
- Unknown target function $f : X \rightarrow Y$
 - Y is discrete valued
- Set of function hypotheses $H = \{ h \mid h : X \rightarrow Y \}$
 - Each hypothesis h is a decision tree
 - Trees sorts x to leaf, which assigns y

Input:

- Training examples $\{ \langle x_{(i)}, y_{(i)} \rangle \}$ of unknown target function f

Output:

- Hypothesis $h \in H$ that best approximates target function f

Top Down Induction of Decision Trees



Node =Root

Main loop:

1. $A \leftarrow$ the "best" decision attribute for next node
2. Assign A as decision attribute for node
3. For each value of A, create new descendant of node
4. Sort training examples to leaf nodes
5. If training examples perfectly classified,
 - a. Then STOP
 - b. Else iterate over new leaf nodes



Entropy

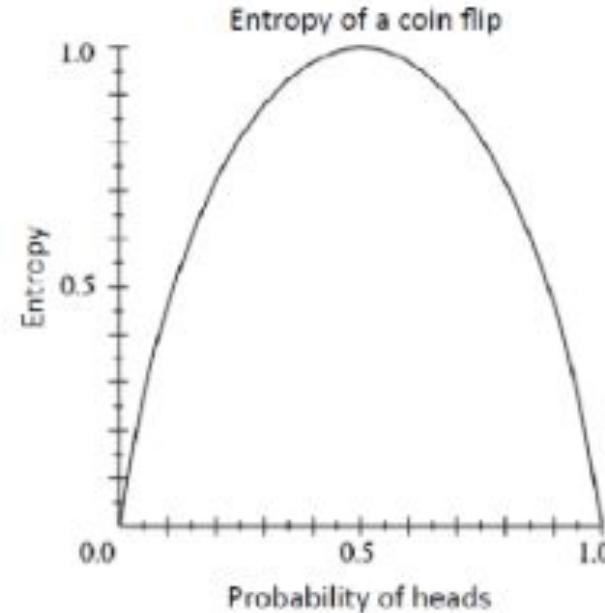


- **Which attribute is best?**
- The entropy -> information content
 - a. More uncertainty -> More entropy [Predictability?]

$$H(X) = - \sum_{i=1}^n P(X = i) \log_2 P(X = i)$$

Result	Prob
H	0.5
T	0.5

Result	Prob
H	0.75
T	0.25



Numerical: $H(Y) = ?$



Y
T
T
T
T
F

$$P(Y=t) = 5/6$$

$$P(Y=f) = 1/6$$

$$\begin{aligned}H(Y) &= - \frac{5}{6} \log_2 \frac{5}{6} - \frac{1}{6} \log_2 \frac{1}{6} \\&= 0.65\end{aligned}$$

Conditional Entropy

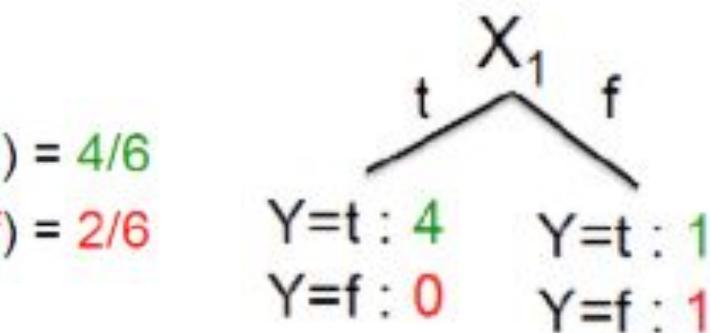


$$H(Y|X) = \sum_{j=1}^v P(X = x_j) \sum_{i=1}^k P(Y = y_i | X = x_j) \log_2 P(Y = y_i | X = x_j)$$

X_1	X_2	Y
T	T	T
T	F	T
T	T	T
T	F	T
F	T	T
F	F	F

$$P(X_1=t) = 4/6$$

$$P(X_1=f) = 2/6$$



$$H(Y|X_1) = - 4/6 (1 \log_2 1 + 0 \log_2 0)$$

$$- 2/6 (1/2 \log_2 1/2 + 1/2 \log_2 1/2)$$

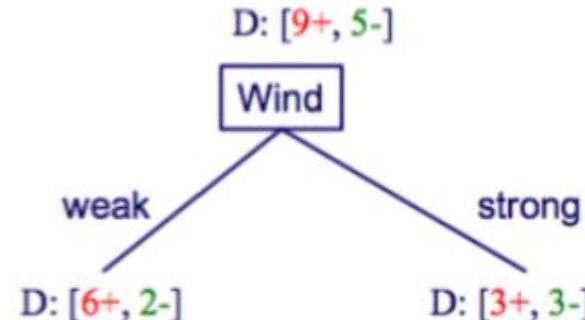
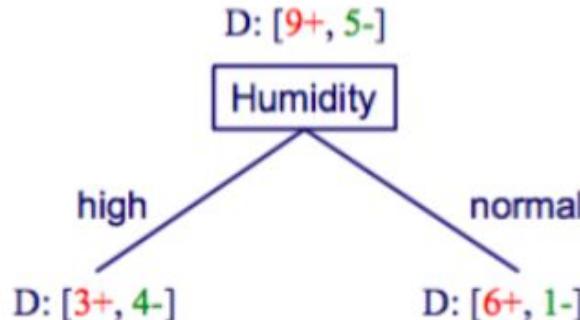
$$= 2/6$$

$$H(Y|X_1) = ?$$

Information Gain: ID3



- Purity (or impurity) is homogeneity (or heterogeneity) of the data.
 - Entropy measures the impurity of the training sample S.
- Information Gain is the expected reduction in entropy after splitting
 - $IG(X) = H(Y) - H(Y|X)$;
 - $IG(X) > 0$; split is preferred



Selecting the next attribute: H, W?



PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

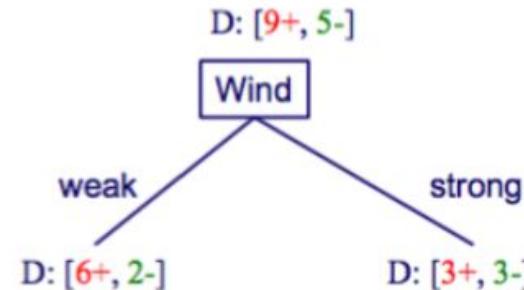
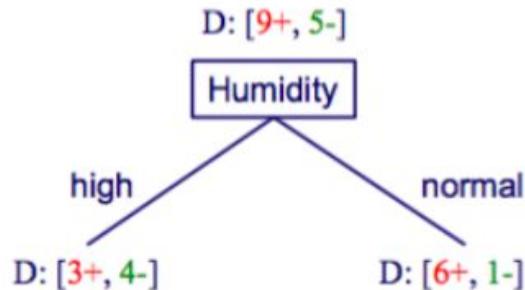
Selecting the next attribute: H, W?



$$H(Y) = - \sum_{i=1}^k P(Y = y_i) \log_2 P(Y = y_i)$$

$$H(Y | X) = - \sum_{j=1}^v P(X = x_j) \sum_{i=1}^k P(Y = y_i | X = x_j) \log_2 P(Y = y_i | X = x_j)$$

$$IG(X) = H(Y) - H(Y | X)$$



Selecting the next attribute: Humidity



- $IG(\text{Humidity}) = H(\text{PlayTennis}) - H(\text{PlayTennis}|\text{Humidity})$
- $H(\text{PlayTennis})$
 $= -(9/14) \cdot \log_2(9/14) - (5/14) \cdot \log_2(5/14) = 0.940$
- $H(\text{PlayTennis}|\text{Humidity})$
 $=$
 $-P(\text{High}) \cdot [P(\text{PlayTennis}|\text{High}) \cdot \log_2 P(\text{PlayTennis}|\text{High}) + P(\sim \text{PlayTennis}|\text{High}) \cdot \log_2 P(\sim \text{PlayTennis}|\text{High})]$
 $-$
 $P(\text{Normal}) \cdot [P(\text{PlayTennis}|\text{Normal}) \cdot \log_2 P(\text{PlayTennis}|\text{Normal}) + P(\sim \text{PlayTennis}|\text{Normal}) \cdot \log_2 P(\sim \text{PlayTennis}|\text{Normal})]$

Selecting the next attribute: Humidity



$$\begin{aligned}H_D(Y \mid \text{high}) &= -\frac{3}{7} \log_2 \left(\frac{3}{7} \right) - \frac{4}{7} \log_2 \left(\frac{4}{7} \right) \\&= 0.985\end{aligned}$$

$$\begin{aligned}H_D(Y \mid \text{normal}) &= -\frac{6}{7} \log_2 \left(\frac{6}{7} \right) - \frac{1}{7} \log_2 \left(\frac{1}{7} \right) \\&= 0.592\end{aligned}$$

$$\begin{aligned}\text{InfoGain}(D, \text{Humidity}) &= 0.940 - \left[\frac{7}{14} (0.985) + \frac{7}{14} (0.592) \right] \\&= 0.151\end{aligned}$$

Selecting the next attribute: Wind



$$H_D(Y \mid \text{weak}) = 0.811$$

$$H_D(Y \mid \text{strong}) = 1.0$$

$$\begin{aligned}\text{InfoGain}(D, \text{Wind}) &= 0.940 - \left[\frac{8}{14}(0.811) + \frac{6}{14}(1.0) \right] \\ &= 0.048\end{aligned}$$

Selecting the next attribute: H, W?



- $IG(\text{Humidity}) = 0.151$
- $IG(\text{Wind}) = 0.048$



Selecting the next attribute: H, W?



- $IG(\text{Humidity}) = 0.151$
- $IG(\text{Wind}) = 0.048$
 - *It is better to split on humidity rather than wind as humidity has a higher information gain.*



Gini Impurity: CART (Classification And Regression Trees)



$$Gini = 1 - \sum_{j=1}^c p_j^2$$

$$I(A) = 1 - P(A_+)^2 - P(A_-)^2$$

$$I(Al) = 1 - P(Al_+)^2 - P(Al_-)^2$$

$$I(Ar) = 1 - P(Ar_+)^2 - P(Ar_-)^2$$

$$GiniGain(A) = I(A) - p_{left}I(Al) - p_{right}I(Ar)$$

Continuous Valued Attributes: Jugaad!



- Create a discrete attribute to test continuous!
- Temperature = 82.5
- Temperature > 70 [T, F]

Temperature	40	48	60	72	80	90
Temp_Jugaad	False	False	False	True	True	True
PlayTennis	No	No	Yes	Yes	Yes	No

Decision Trees will Overfit!



- Standard decision trees have low bias.
 - Training set error is almost zero!
 - High variance
 - Must introduce some bias towards simpler trees
- Pruning: strategies for picking simpler trees
 - Pre-pruning
 - Fixed depth
 - Fixed number of leaves
 - Post-pruning



References



1. Constructing optimal binary decision trees is NP-complete." Information Processing Letters 5.1 (1976): 15-17.
2. Entropy: <https://www.cs.utexas.edu/~byoung/cs361/lecture32.pdf>
3. http://www.cs.cmu.edu/~tom/10701_sp11/slides/DTreesAndOverfitting-1-11-2011_final.pdf
4. <https://www.ke.tu-darmstadt.de/lehre/archiv/wso809/mldm/dt.pdf>
5. [Theoretical comparison between the Gini Index and Information Gain criteria](#)
- 6.



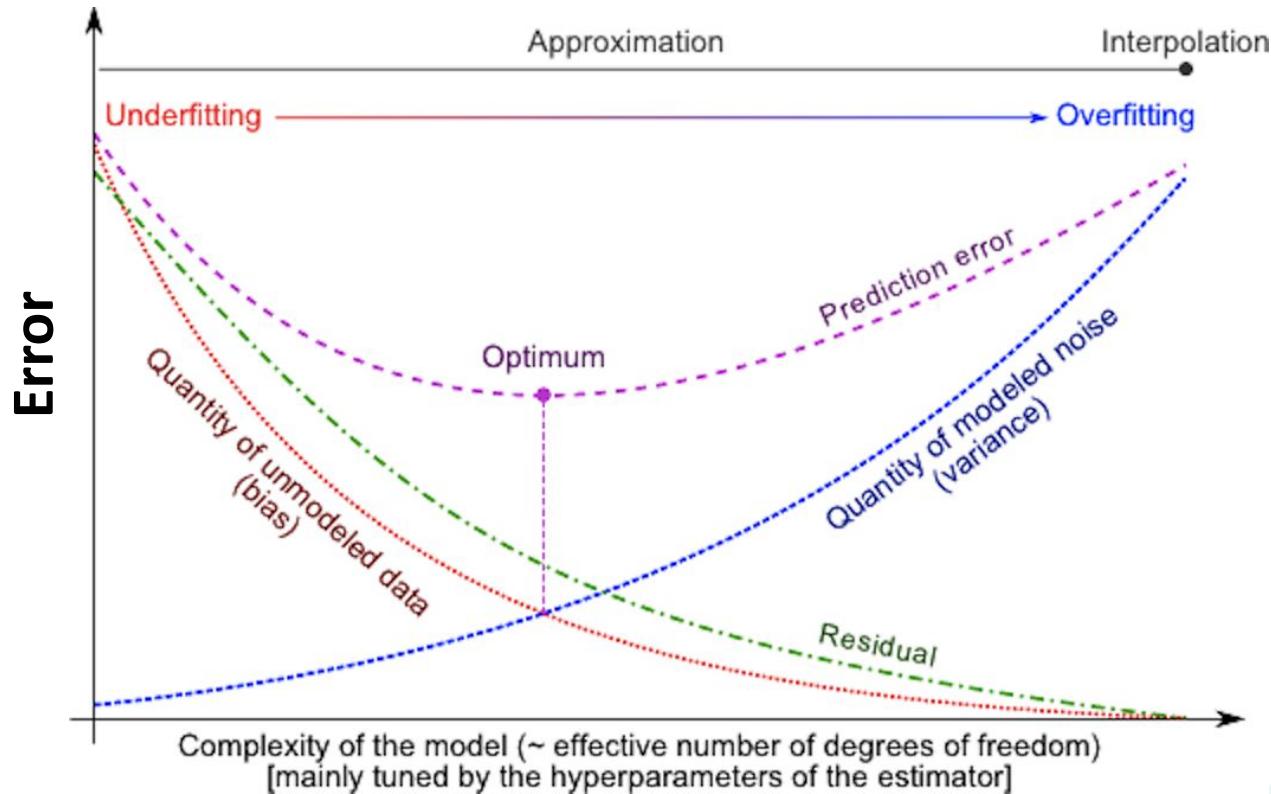
Random Forests



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



Bias vs. Variance Analysis



Reduce Variance Without Increasing Bias



- Averaging reduces variance:

$$Var(\bar{X}) = \frac{Var(X)}{N}$$

- Average models to reduce model variance
- Problem: Only one training set
 - Where do multiple models come from?
- Ensemble learning
 - Combining *weak classifiers* (of the same type) in order to produce a strong classifier
 - Condition: diversity among the weak classifiers
- *Weak classifiers*: only need to be better than random guess

Decision Trees -> Random Forests



- DT are non-flexible -> Inaccurate
 - Performance on unseen data suffers -> High Variance
- Definition: Random Forests
 - Collection of unpruned CARTs
 - Rule to combine individual tree decisions
- Ensemble learning: Two ways to introduce randomness/diversity
 - “Bagging” and “Random input vectors”
 - Bagging method: each tree is grown using a bootstrap sample of training data
 - Random vector method: At each node, best split is chosen from a random sample of m attributes instead of all attributes.

Random Forests

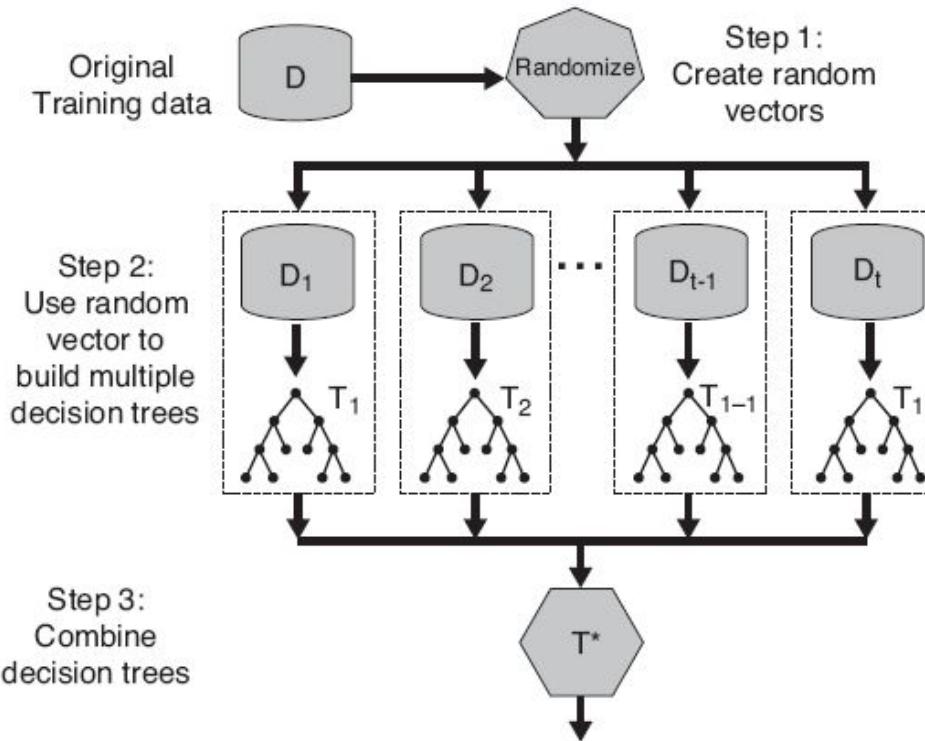


Figure 5.40. Random forests.

Bagging



- L : original learning set composed of n samples
- Generate K learning sets $L_k \dots$
 - ... composed of m samples, $m \leq n, \dots$
 - ... obtained by uniform sampling with replacement from L
 - In consequences, L_k may contain repeated samples

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

0.632 Bootstrap



- $m = n$
- A particular training data has a probability of $(1-1/n)$ of not being picked
- Thus its probability of ending up in the test data (not selected) is, when picking n data points :

$$\left(1 - \frac{1}{n}\right)^n \approx e^{-1} = 0.368$$

- -> Training data will contain approximately 63.2% of the instances
- Out of Bag (OOB) samples-> Out-Of-Bag Error

Out-of-bag error



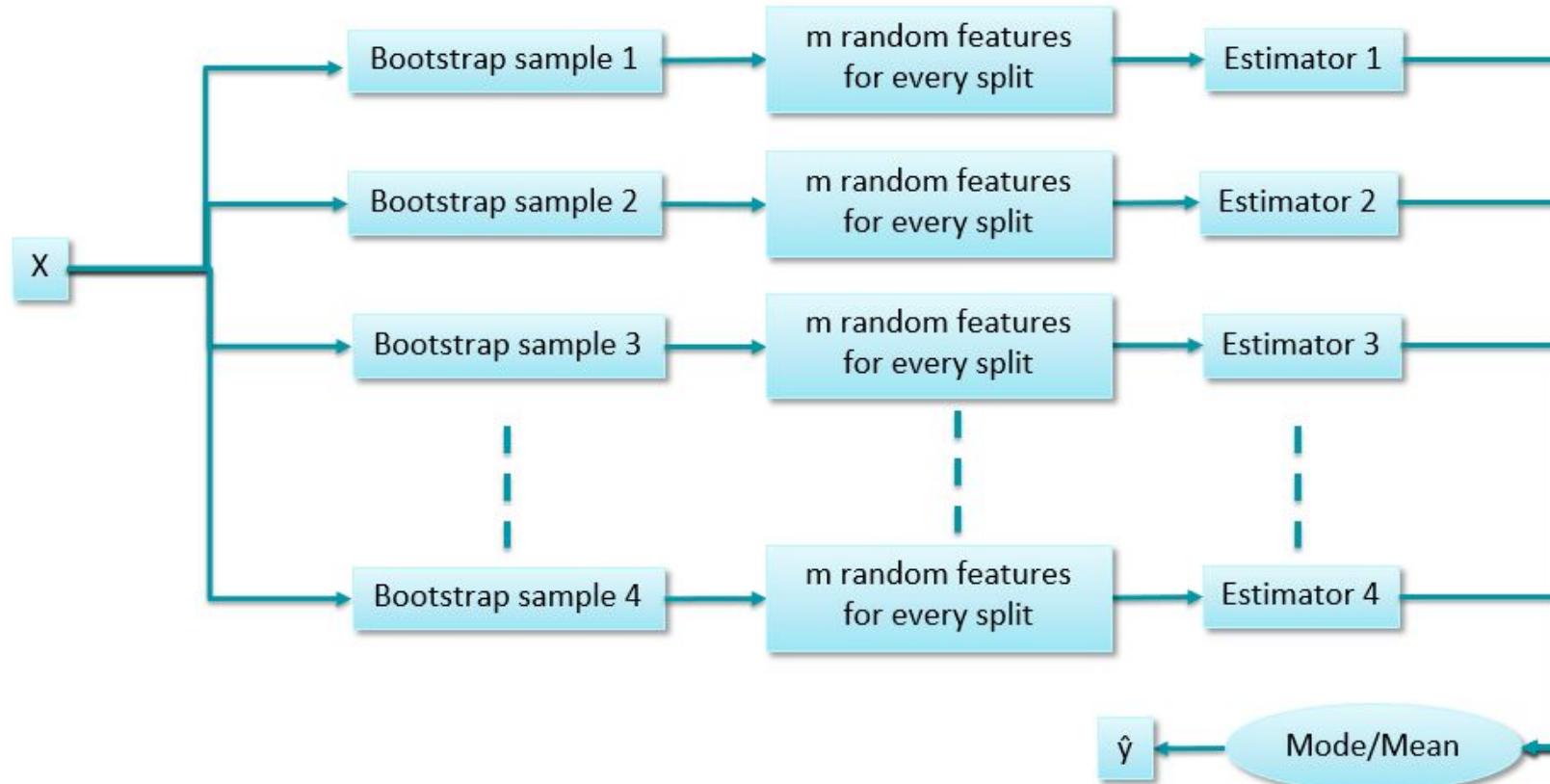
- For each sample S of the OOB set
 - Predict the class of S with OOB classifiers that does not contain the sample S
 - Error = is prediction correct?
- Out-of-bag error = average over all samples of S
- Provides an estimation of the generalization error
 - Can be used to decide when to stop adding trees to the forest
 - As accurate as using a test set of the same size as the training set.
 - Therefore, using the out-of-bag error estimate removes the need for a set aside test set.

Random forest > Bagging > Aggregation

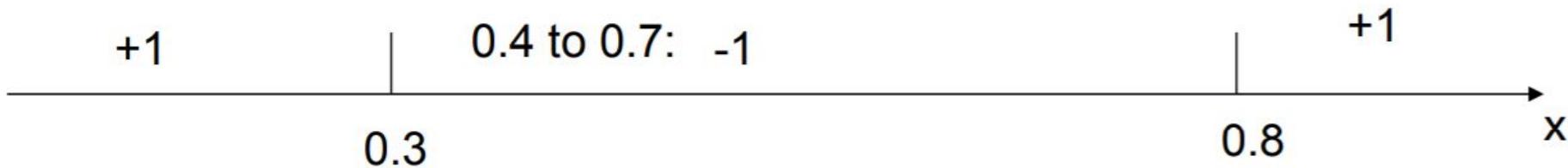


- Bagging: Bootstrap aggregation
- Technique of ensemble learning
 - To avoid over-fitting
 - Important since trees are unpruned
 - To improve stability and accuracy
- Idea
 - Create N bootstrap samples $\{S_1, \dots, S_N\}$ of S as follows:
 - For each S_i randomly draw $|S|$ examples from S with replacement
 - For each $i = 1, \dots, N$, $h_i = \text{Learn}(S_i)$
 - Output $H = \langle \{h_1, \dots, h_N\}, \text{majorityVote} \rangle$
 - Use average or majority voting to aggregate results

Random forest > Bagging > Aggregation



Bagging Example -1



Goal: Find a collection of 10 simple thresholding classifiers that collectively can classify correctly.

- Each simple (or weak) classifier is:
 $(x \leq K \text{ class} = +1 \text{ or } -1 \text{ depending on which value yields the lowest error})$

Bagging Example - I



Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9	$x \leq 0.35 \Rightarrow y = 1$
y	1	1	1	1	-1	-1	-1	-1	1	1	$x > 0.35 \Rightarrow y = -1$

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1	$x \leq 0.65 \Rightarrow y = 1$
y	1	1	1	-1	-1	1	1	1	1	1	$x > 0.65 \Rightarrow y = 1$

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9	$x \leq 0.35 \Rightarrow y = 1$
y	1	1	1	-1	-1	-1	-1	-1	1	1	$x > 0.35 \Rightarrow y = -1$

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9	$x \leq 0.3 \Rightarrow y = 1$
y	1	1	1	-1	-1	-1	-1	-1	1	1	$x > 0.3 \Rightarrow y = -1$

Bagging Example - II



Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

$$\begin{aligned}x \leq 0.35 &\Rightarrow y = 1 \\x > 0.35 &\Rightarrow y = -1\end{aligned}$$

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

$$\begin{aligned}x \leq 0.75 &\Rightarrow y = -1 \\x > 0.75 &\Rightarrow y = 1\end{aligned}$$

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

$$\begin{aligned}x \leq 0.75 &\Rightarrow y = -1 \\x > 0.75 &\Rightarrow y = 1\end{aligned}$$

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$$\begin{aligned}x \leq 0.75 &\Rightarrow y = -1 \\x > 0.75 &\Rightarrow y = 1\end{aligned}$$

Bagging Example - III



Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

$x \leq 0.75 \Rightarrow y = -1$
 $x > 0.75 \Rightarrow y = 1$

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

$x \leq 0.05 \Rightarrow y = -1$
 $x > 0.05 \Rightarrow y = 1$

Bagging Example - IV



Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Method for Growing the tree



- Fix a $m \leq M$. At each node
- Method 1
 - Choose m attributes randomly, compute their information gains, and choose the attribute with the largest gain to split
- Method 2: (When M is not very large):
 - Select L of the attributes randomly. Compute a linear combination of the L attributes using weights generated from $[-1,+1]$ randomly. That is, new $A = \text{Sum}(W_i^*M_i)$, $i=1..L$.

Method for Growing the tree



-
- Method 3:
 - Compute the information gain of all M attributes.
Select the top m attributes by information gain.
Randomly select one of the m attributes as the splitting node.

Value of m



- For classification, the default value for m is $\lfloor \sqrt{M} \rfloor$ and the minimum node size is one.
- For regression, the default value for m is $\lfloor M/3 \rfloor$ and the minimum node size is five.



Reducing Variance



- The variance of B i.i.d. random variables $\frac{\sigma^2}{B}$
 - B = number of trees, σ^2 = variance of each individual tree
- If the variables are simply i.d. (identically distributed, but not necessarily independent) with positive pairwise correlation ρ the variance of the ensemble is $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$
- The variance of an ensemble is strictly smaller than the variance of an individual model.

Reducing Variance



$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

- As B increases, the second term disappears
 - Size of the correlation of pairs of bagged trees limits the benefits of averaging
- To improve the variance reduction of bagging by reducing the correlation ρ between the trees, without increasing the variance too much
 - Random selection of the input variables during tree growing
 - Specifically, before each split, select $m \leq p$ of the input variables at random as candidates for splitting

Reducing Bias and Variance



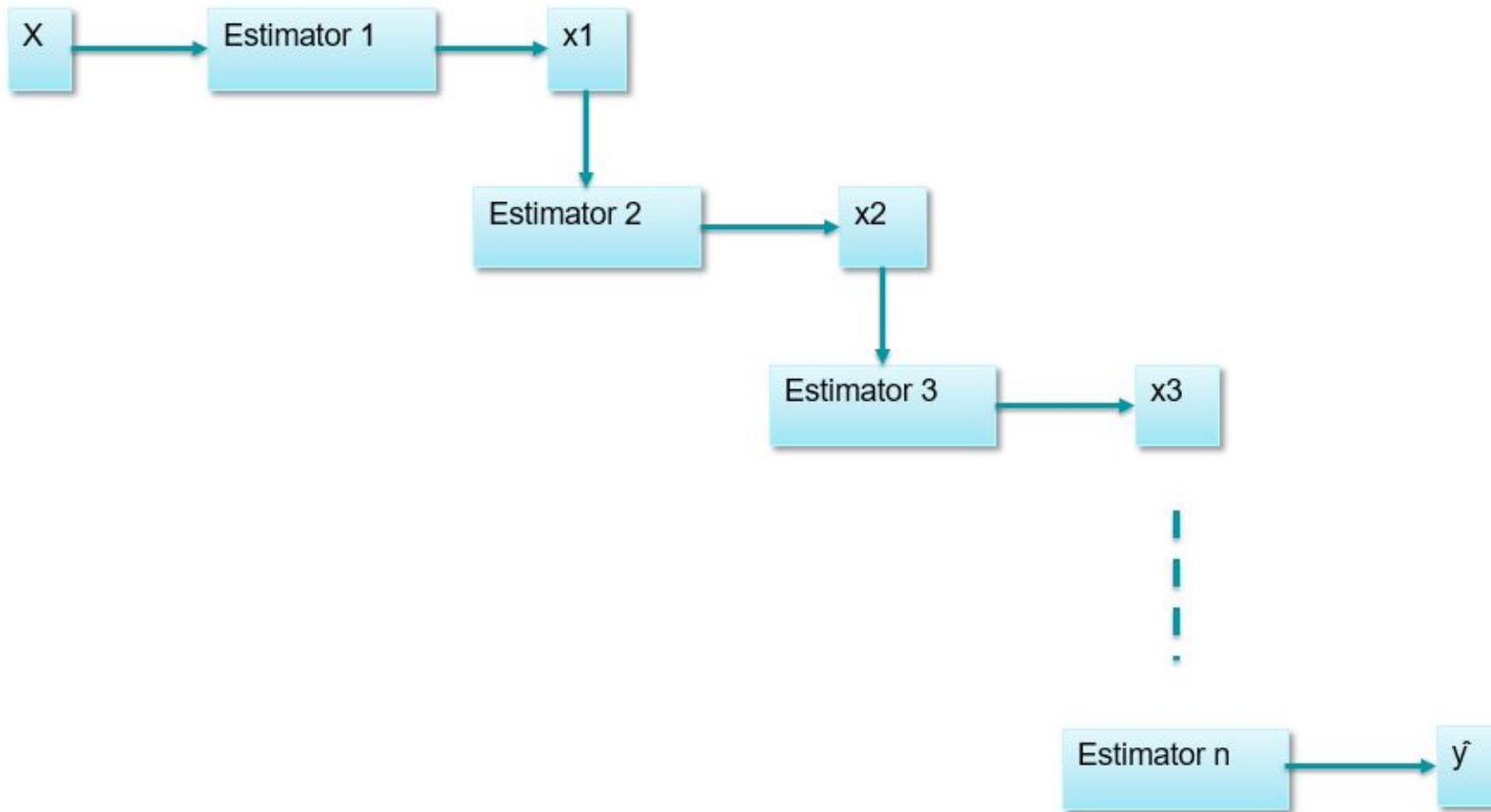
- Bagging: Each individual classifier is independent
 - Bagging has little effect on bias
- Each tree generated in bagging is identically distributed (i.d.), the expectation of an average of B such trees is
 - the same as the expectation of any one of them
 - The bias of bagged trees is the same as that of the individual trees
- Can we average and reduce bias?
 - YES
 - Boosting
- In Boosting the trees are grown in an adaptive way to remove bias, and hence are not i.d.

Boosting



- Boosting is iterative and adaptive:
 - Looks at the errors from previous classifiers to decide what to focus on for the next iteration
 - Successive classifiers depend on their predecessors
 - Key idea: place more weight on “hard” examples (i.e., instances that were misclassified on previous iterations)
 - Records that are classified correctly will have their weights decreased
- Adaboost – popular boosting algorithm

Boosting



Bagging and Boosting Summary



Bagging

- Resample data points
- Weight of each classifier is the same
- Only variance reduction
- Robust to noise and outliers

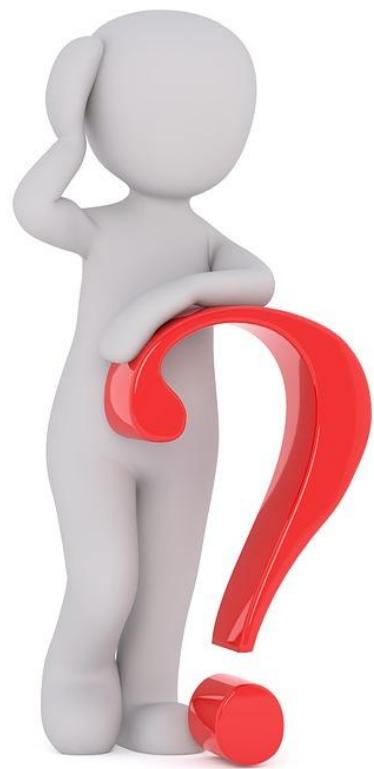
Boosting

- Reweighting data points(modify data distribution)
- Weight of classifier vary depending on accuracy
- Reduces both bias and variance
- Can hurt performance with noise and outliers

References



1. The Elements of Statistical Learning, Chapter 15
2. Efron, B. and R. Tibshirani (1997), "Improvements on Cross-Validation: The .632+ Bootstrap Method," Journal of the American Statistical Association Vol. 92, No. 438. (Jun), pp. 548-560
3. <https://perso.math.univ-toulouse.fr/motimo/files/2013/07/random-for est.pdf>
4. <https://stat.ethz.ch/education/seminars/ss2012/ams/slides/v10.2.pdf>
5. [StatQuest: Random Forests Part 1 - Building, Using and Evaluating](#)
 - a. Look at Part 2 also.



Perceptron



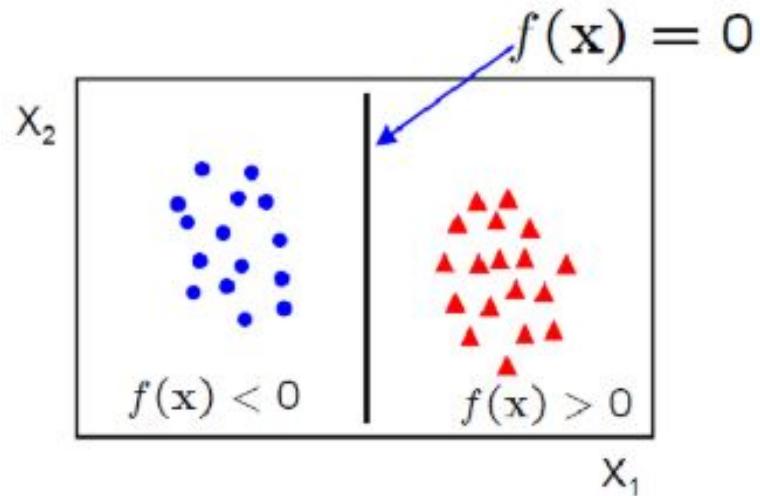
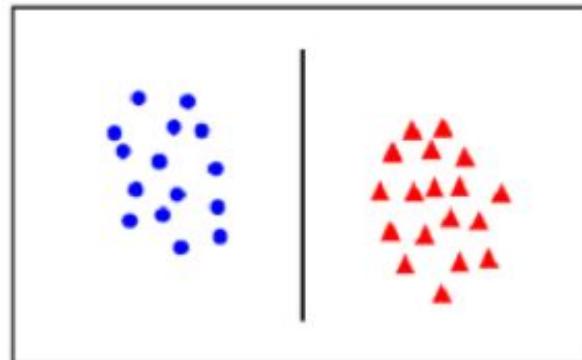
INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



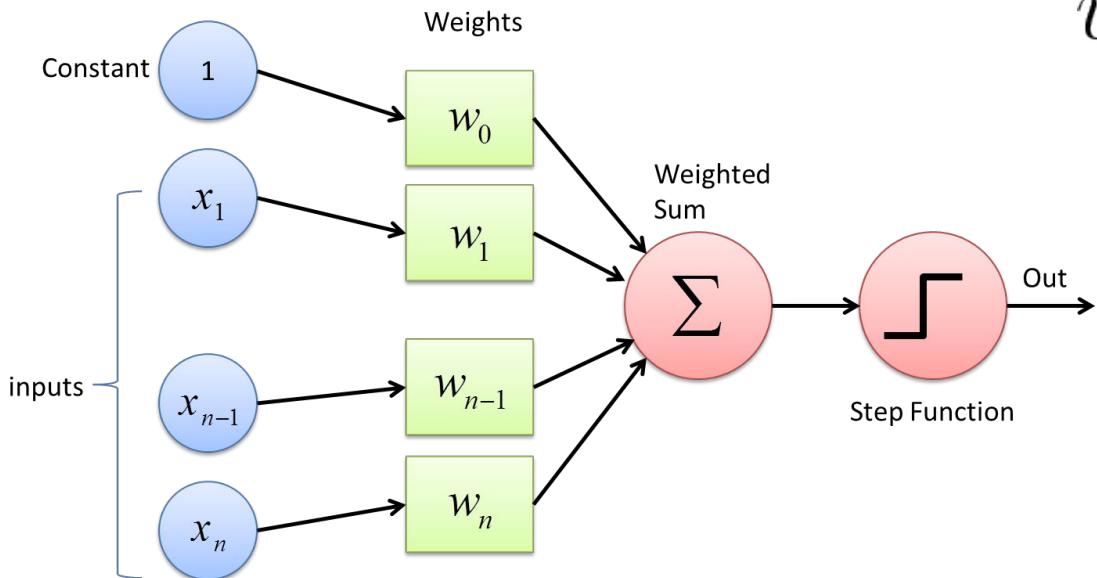
Perceptron



- **Input:** $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ s.t., $x_i \in X, y_i \in Y$ where $X \in \mathbb{R}$, $Y \in \{-1, 1\}$
- Use a function to map real numbers to $\{-1, 1\}$



Perceptron: Binary Classification



$$v = \sum_{i=1}^m w_i x_i + b$$

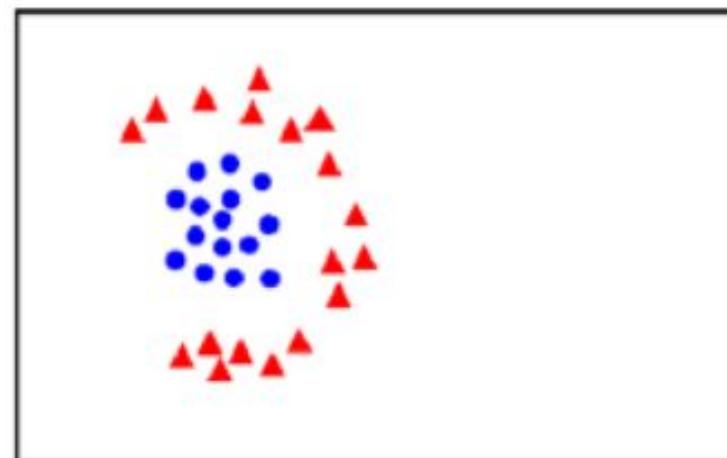
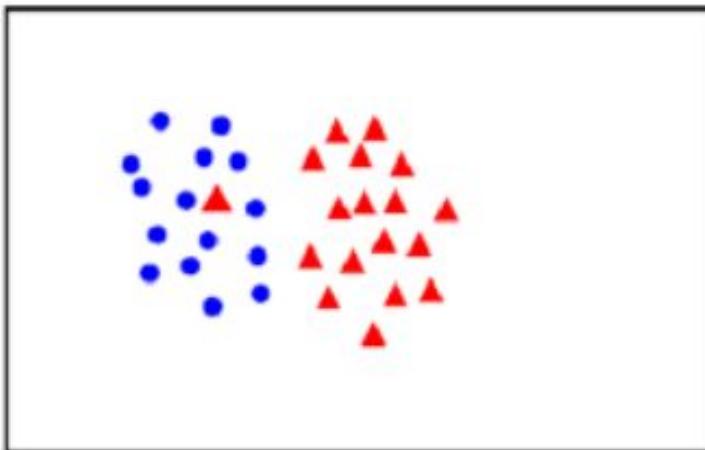
$$\sum_{i=1}^m w_i x_i + b = 0$$

$$sgn(v) = \begin{cases} +1 & \text{if } v \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

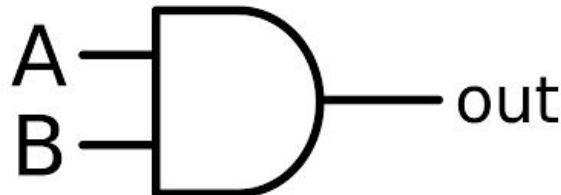
Separability: Linear vs Non-Linear



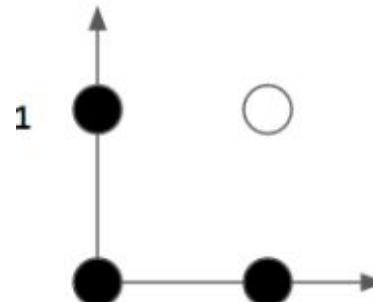
- **Assumption:** There exists a hyperplane!



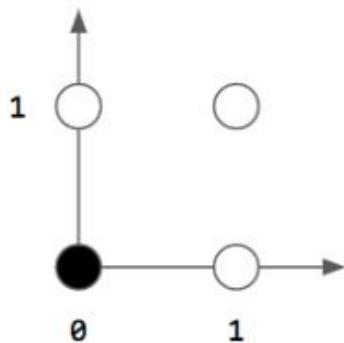
Logical Operations



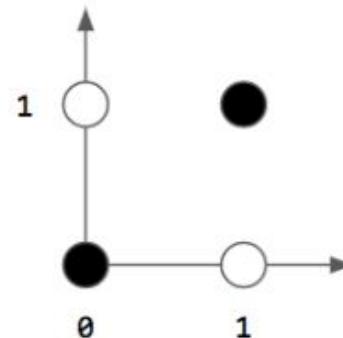
AND



OR



XOR

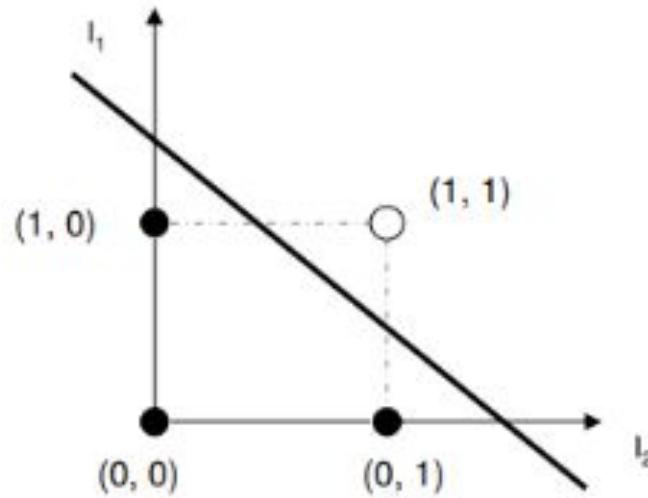


Example: AND Gate



$$x_1 + x_2 - 1.5 = 0$$
$$w_1 = w_2 = 1; b = -1.5$$

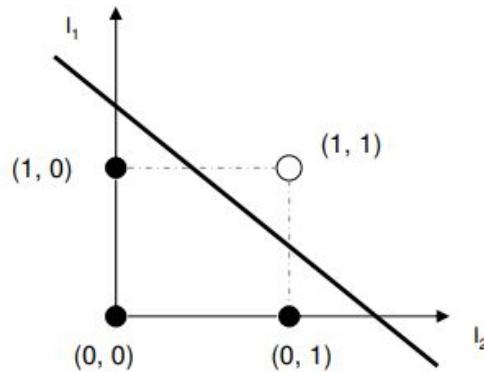
1. $x_1 = 0, x_2 = 0$
a. $0 + 0 - 1.5 = -1.5$
2. $x_1 = 0, x_2 = 1$
a. $0 + 1 - 1.5 = -0.5$
3. $x_1 = 1, x_2 = 0$
a. $1 + 0 - 1.5 = -0.5$
4. $x_1 = 1, x_2 = 1$
a. $1 + 1 - 1.5 = 0.5$



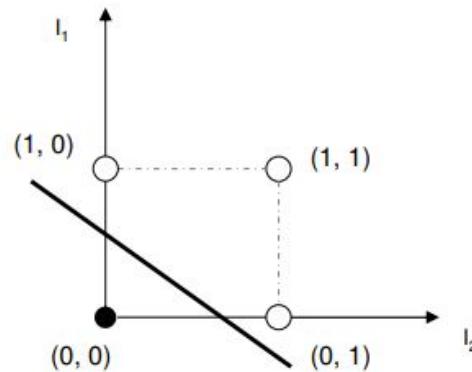
Hyperplane for Logical Operations



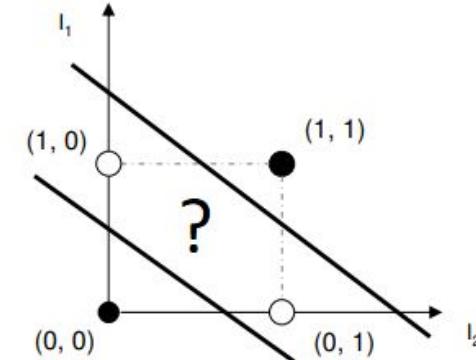
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



Learning



- Correct Classification: $y_i f(x_i) > 0$
 - $f(x_i) = b + w_1 x_1 + w_2 x_2 + \dots + w_n x_n > 0$ and belongs to C_1
 - $f(x_i) = b + w_1 x_1 + w_2 x_2 + \dots + w_n x_n < 0$ and belongs to C_2
 - $w(i+1) = w(i)$
- Incorrect Classification: $y_i f(x_i) < 0$
 - $f(x_i) = b + w_1 x_1 + w_2 x_2 + \dots + w_n x_n < 0$ and belongs to C_1
 - $w(i+1) = w(i) + \Delta$
 - $f(x_i) = b + w_1 x_1 + w_2 x_2 + \dots + w_n x_n > 0$ and belongs to C_2
 - $w(i+1) = w(i) - \Delta$

Perceptron Learning Algorithm



- **Input:** Training examples $\{x_i, y_i\}_{i=1 \text{ to } n}$
- Initialize w and b as zero or randomly
- While !converged do
 - #Loop through the samples
 - *for j = 1 to n do*
 - #Compare the true label and the prediction
 - $\text{error}_j = y_j - \varphi(w^T x_j + b)$
 - #If the model wrongly predicts the class, update the weight and the bias
 - If $\text{error} \neq 0$
 - #Update the weight
 - $W = w + \text{error}_j \times x_j$
 - #Update the bias
 - $B = b + \text{error}_j$
 - Test for convergence
 - **Output:** Set of weights w and bias b for the perceptron

Perceptron Learning Algorithm



Let's follow the steps manually for the AND gate.

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$0 \rightarrow -1$
 $1 \rightarrow 1$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 0$ and $b = 0$
 - Let's calculate error and compare the true label and the prediction
 - $sample1_prediction = 1$
 - $sample1_error = -1 - 1 = -2$

Iteration=1

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

-1
-1
-1

$$\text{error} = y - \varphi(w^T x + b)$$

$$\Phi = \text{sgn function}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 0$ and $b = 0$
 - Let's calculate error and compare the true label and the prediction
 - $sample1_prediction = 1$
 - $sample1_error = -1 - 1 = -2$
 - If $error \neq 0$
 - Updating the weights
 - $W_1 = 0 + (-2)*0 = 0$
 - $W_2 = 0 + (-2)*0 = 0$
 - Updating the bias
 - $B = 0 + (-2) = -2$
 - $W_1, W_2 = 0$ and $b = -2$

Iteration=1

AND		
I ₁	I ₂	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 0$ and $b = -2$
 - Let's calculate error and compare the true label and the prediction
 - $sample2_prediction = -1$
 - $sample2_error = -1 + 1 = 0$
 - If $error \neq 0$
 - Updating the weights
 - Updating the bias
 - $W_1, W_2 = 0$ and $b = -2$

Iteration=1

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

-1
-1
-1

$$\begin{aligned}error &= y - \varphi(w^T x + b) \\W &= w + error \otimes x \\B &= b + error\end{aligned}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 0$ and $b = -2$
 - Let's calculate error and compare the true label and the prediction
 - $sample3_prediction = -1$
 - $sample3_error = -1 + 1 = 0$
 - If $error \neq 0$
 - Updating the weights
 - Updating the bias
 - $W_1, W_2 = 0$ and $b = -2$

Iteration=1

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

-1
-1
-1

$$\begin{aligned}error &= y - \varphi(w^T x + b) \\W &= w + error \otimes x \\B &= b + error\end{aligned}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 0$ and $b = -2$
 - Let's calculate error and compare the true label and the prediction
 - $sample4_prediction = -1$
 - $sample4_error = 1 + 1 = 2$
 - If $error \neq 0$
 - Updating the weights
 - $W1 = 0 + (2)*1 = 2$
 - $W2 = 0 + (2)*1 = 2$
 - Updating the bias
 - $B = -2 + (2) = 0$
 - $W_1, W_2 = 2$ and $b = 0$

Iteration=1

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 2$ and $b = 0$
 - Let's calculate error and compare the true label and the prediction
 - $sample1_prediction = 1$
 - $sample1_error = -1 - 1 = -2$
 - If $error \neq 0$
 - Updating the weights
 - $W_1 = 2 + (-2)*0 = 2$
 - $W_2 = 2 + (-2)*0 = 2$
 - Updating the bias
 - $B = 0 + (-2) = -2$
 - $W_1, W_2 = 2$ and $b = -2$

Iteration=2

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 2$ and $b = -2$
 - Let's calculate error and compare the true label and the prediction
 - $sample2_prediction = 1$
 - $sample2_error = -1 - 1 = -2$
 - If $error \neq 0$
 - Updating the weights
 - $W1 = 2 + (-2)*0 = 2$
 - $W2 = 2 + (-2)*0 = 2$ ← Error is made intentionally (try correcting it)
 - Updating the bias
 - $B = -2 + (-2) = -4$
 - $W_1, W_2 = 2$ and $b = -4$

Iteration=2

AND		
I ₁	I ₂	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 2$ and $b = -4$
 - Let's calculate error and compare the true label and the prediction
 - $sample3_prediction = -1$
 - $sample3_error = -1 + 1 = 0$
 - If $error \neq 0$
 - Updating the weights
 - Updating the bias
 - $W_1, W_2 = 2$ and $b = -4$

Iteration=2

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 2$ and $b = -4$
 - Let's calculate error and compare the true label and the prediction
 - sample4_prediction = 1
 - sample4_error = 1 -1 = 0
 - If error != 0
 - Updating the weights
 - Updating the bias
 - $W_1, W_2 = 2$ and $b = -4$

Iteration=2

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

-1
-1
-1

$$\begin{aligned} \text{error} &= y - \varphi(w^T x + b) \\ W &= w + \text{error} \otimes x \\ B &= b + \text{error} \end{aligned}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 2$ and $b = -4$
 - Let's calculate error and compare the true label and the prediction
 - $sample1_prediction = -1$
 - $sample1_error = -1 + 1 = 0$
 - If $error \neq 0$
 - Updating the weights
 - Updating the bias
 - $W_1, W_2 = 2$ and $b = -4$

Iteration=3

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 2$ and $b = -4$
 - Let's calculate error and compare the true label and the prediction
 - $sample2_prediction = -1$
 - $sample2_error = -1 + 1 = 0$
 - If $error \neq 0$
 - Updating the weights
 - Updating the bias
 - $W_1, W_2 = 2$ and $b = -4$

Iteration=3

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 2$ and $b = -4$
 - Let's calculate error and compare the true label and the prediction
 - $sample3_prediction = -1$
 - $sample3_error = -1 + 1 = 0$
 - If $error \neq 0$
 - Updating the weights
 - Updating the bias
 - $W_1, W_2 = 2$ and $b = -4$

Iteration=3

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$

Perceptron Learning Algorithm



- **Input:** Training examples for the AND gate problem?
 - Input is clear.
 - Map the out to -1 and 1 s.t., 1 maps to 1 and 0 maps to -1
- Initialize w and b as zero.
 - $W_1, W_2 = 2$ and $b = -4$
 - Let's calculate error and compare the true label and the prediction
 - sample4_prediction = 1
 - sample4_error = 1 - 1 = 0
 - If error != 0
 - Updating the weights
 - Updating the bias
 - $W_1, W_2 = 2$ and $b = -4$
- **Output:** $W_1, W_2 = 0.5$ and $b = -1$

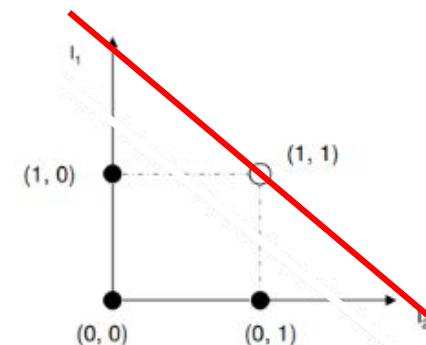
Iteration=3

AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

$$\text{error} = y - \varphi(w^T x + b)$$

$$W = w + \text{error} \otimes x$$

$$B = b + \text{error}$$



Perceptron Convergence Theorem



$$\mathbf{w}(n + 1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T \mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_1$$

$$\mathbf{w}(n + 1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T \mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_2$$

$$\mathbf{w}(n + 1) = \mathbf{w}(n) - \eta(n) \mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n) \mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_2$$

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \eta(n) \mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n) \mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathcal{C}_1$$

Assumptions:

- Learning rate $\eta = 1$
- Initial Condition $w(0) = o$

Perceptron Convergence Theorem



- Misclassification for $x(1), x(2) \dots, x(n) \in C_1$
 - $w(n+1)$
 - $= w(n) + x(n)$
 - $w(0) = o;$
 - $w(1) = x(o)$
 - $w(2) = w(1) + x(1)$
 - $w(n+1) = x(o) + x(1) + x(2) + \dots + x(n) \text{ ---- (I)}$
- Since C_1 and C_2 are linearly separable, there exists optimal w_o such that $w_o^T x(n) > o$ for $x(1), x(2) \dots, x(n) \in C_1$
 - Let a be a positive number
 - $\alpha = \min_{x(n) \in C_1} w_o^T x(n)$

Perceptron Convergence Theorem



- Multiplying w_o^T both the sides of the equation (I)
 - $w_o^T w(n+1) = w_o^T x(1) + w_o^T x(2) + \dots + w_o^T x(n)$
 - $w_o^T w(n+1) \geq n\alpha$
- *Cauchy-Schwarz inequality for two vectors u, v*
 - $\|u\|^2 \|v\|^2 \geq [uv]^2$
 - $\|w_o^T\|^2 \|w(n+1)\|^2 \geq [w_o^T w(n+1)]^2$
- $\|w_o^T\|^2 \|w(n+1)\|^2 \geq n^2 \alpha^2$
 - $\|w(n+1)\|^2 \geq n^2 \alpha^2 / \|w_o^T\|^2 \dots\dots\dots (II)$



Perceptron Convergence Theorem



- $w(k+1) = w(k) + x(k)$, $k = 1, 2, \dots, n$ and $x(k) \in C_1$
- By taking the squared Euclidean norm
 - $\|w(k+1)\|^2 = \|w(k)\|^2 + \|x(k)\|^2 + 2w^T(k)x(k)$
- Misclassification for $x(1), x(2), \dots, x(n) \in C_1$ i.e. $w^T(k)x(k) < 0$
 - $\|w(k+1)\|^2 \leq \|w(k)\|^2 + \|x(k)\|^2$
- $\|w(k+1)\|^2 - \|w(k)\|^2 \leq \|x(k)\|^2$ for $k = 1, 2, \dots, n$
- $w(0) = 0$

$$\|w(n+1)\|^2 \leq \sum_{i=1}^n \|x(k)\|^2$$

$$\|w(n+1)\|^2 \leq n\beta \quad \beta = \max_{x(n) \in C_1} \|x(k)\|^2$$

Perceptron Convergence Theorem



- Conflict

$$\|\mathbf{w}(n + 1)\|^2 \geq \frac{n^2 \alpha^2}{\|\mathbf{w}_0\|^2}$$

$$\|\mathbf{w}(n + 1)\|^2 \leq \sum_{k=1}^n \|\mathbf{x}(k)\|^2 \leq n\beta$$

- n cannot be larger than some value n_{max} for which both are satisfied:

-

$$\frac{n_{max}^2 \alpha^2}{\|\mathbf{w}_0\|^2} = n_{max} \beta \quad n_{max} = \frac{\beta \|\mathbf{w}_0\|^2}{\alpha^2}$$



Perceptron Convergence Theorem



- Weight update algorithm must terminate after n_{max} iterations.
 - If the data are linearly separable, perceptron is guaranteed to converge
- There is no unique solution for w_o (optimal weights) and n_{max} (maximum number of iterations).
- Fixed Increment Convergence Theorem:
 - For some $n_o \leq n_{max}$, the perceptron converges such that $w(n_o) = w(n_o + 1) = w(n_o + 2) = \dots$
 - n_o : optimal number of iterations



Loss: Perceptron and Logistic Regression



$$\mathcal{L}_{\text{perc}}(x, y) = \begin{cases} 0 & \text{if } y\Phi(w^T x) > 0 \\ -y\Phi(w^T x) & \text{if } y\Phi(w^T x) \leq 0 \end{cases}$$

$\Phi = \text{sgn function}$

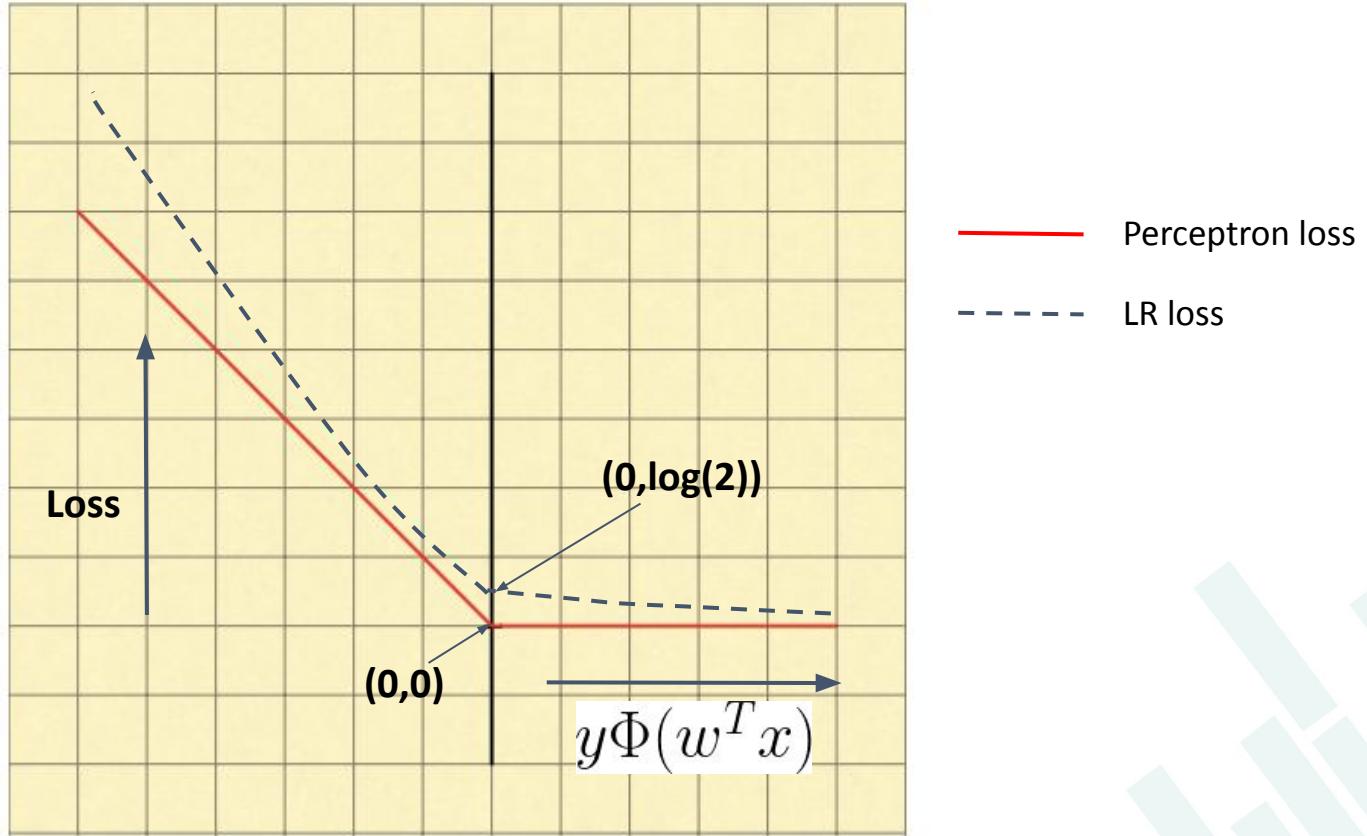
Loss function for perceptron

$$\mathcal{L}_{\text{lr}}(x, y) = \begin{cases} -y\Phi(w^T x) + \log(1 + e^{y\Phi(w^T x)}) & \text{if } y = +1 \text{ (positive)} \\ \log(1 + e^{-y\Phi(w^T x)}) & \text{if } y = -1 \text{ (negative)} \end{cases}$$

$\Phi = \text{sigmoid function}$

Loss function for logistic regression

Loss: Perceptron and Logistic Regression



References



-
1. <https://www.acm.org/media-center/2019/march/turing-award-2018>
 2. Chapter 3, Neural Networks: A Comprehensive Foundation (2nd Edition) 2nd Edition by Simon Haykin





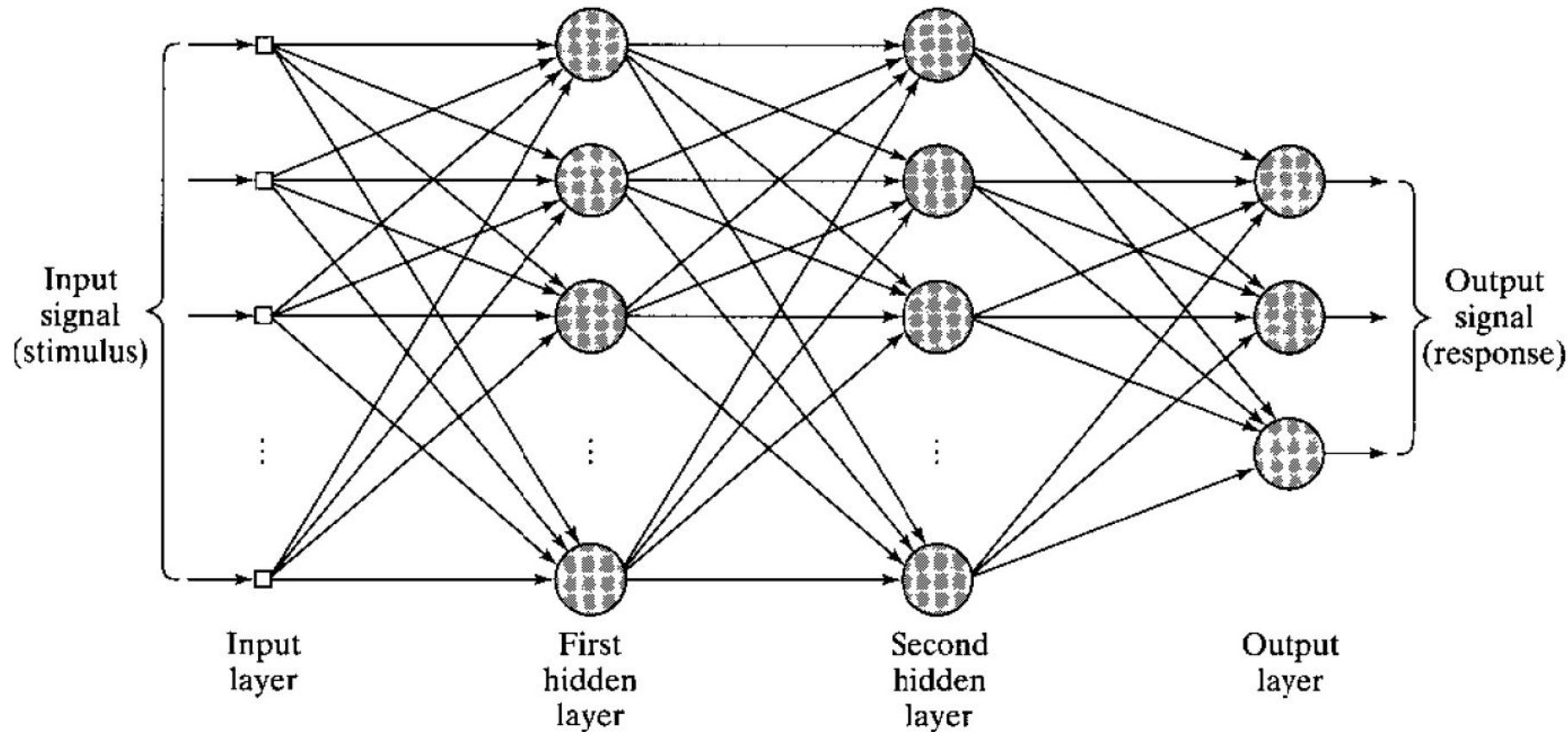
Multilayer Perceptron



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



Multilayer Perceptron



Basic Features



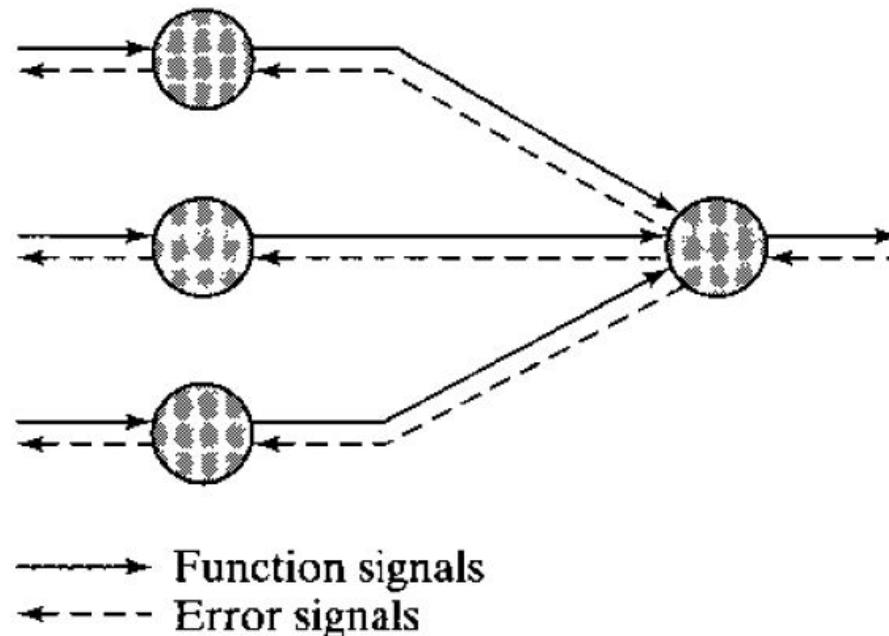
-
- 1. Non-linear activation function that is differentiable
 - 2. Hidden Layers
 - a. Not part of the input or the output of the network.
 - 3. High degree of connectivity: Fully Connected
 - a. The extent of the connections is determined by synaptic weights.
-
- Theoretical Analysis is difficult
 - Learning process is harder to visualize



Phases of MLP



1. Forward Phase: Input/Function Signals
2. Backward Phase: Error Signals



Computations of Hidden/Output Neuron



-
1. The computation of the function signal appearing at the output of each neuron
 2. The computation of an estimate of the gradient vector which is needed for the backward pass.

Hidden Neuron Functions:

- Act as *feature detectors*
- They gradually discover the salient features that characterize the training data
 - Nonlinear transformation: input data space -> feature space

Credit-Assignment Problem

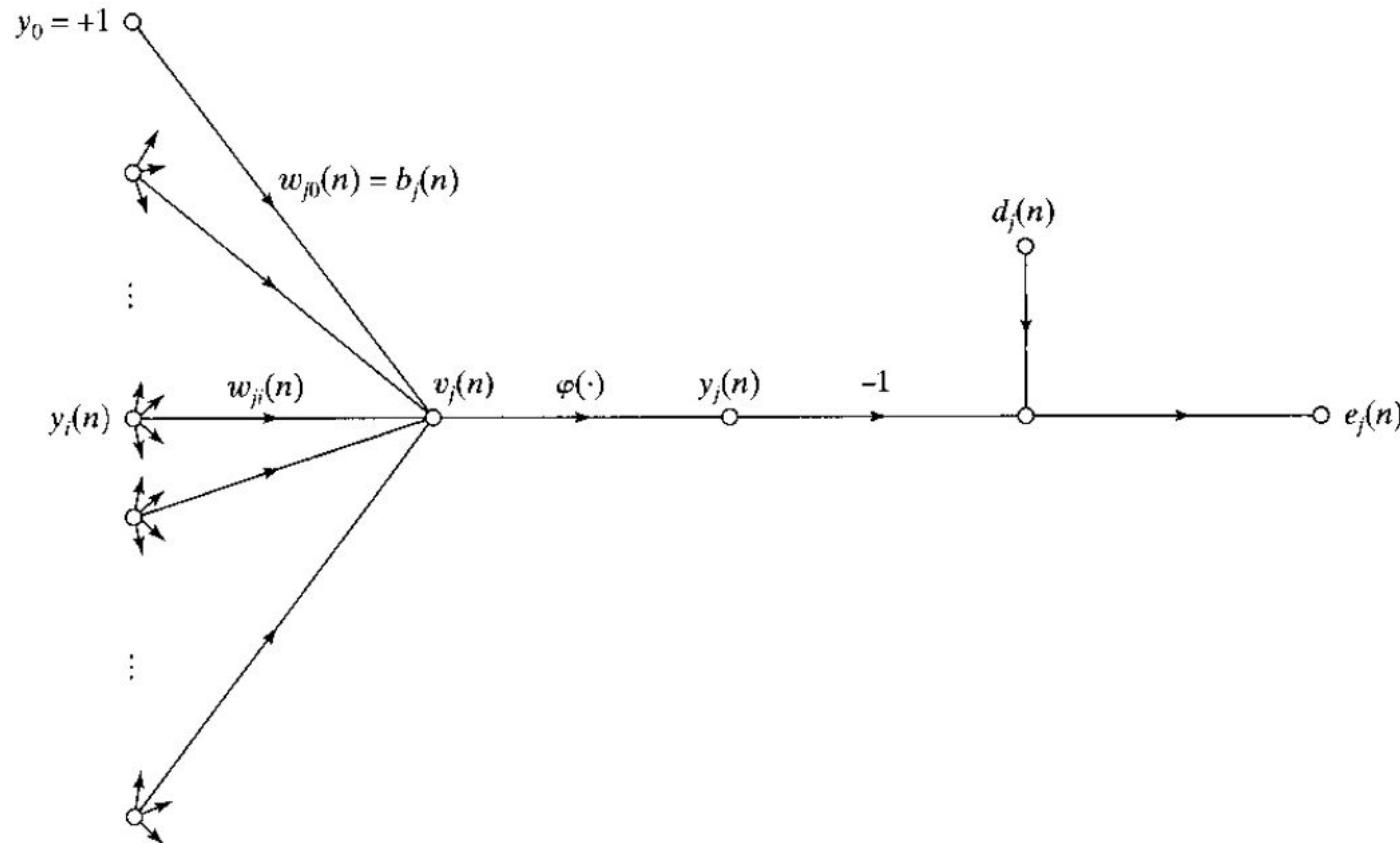


The problem of assigning *credit* or *blame* for overall outcomes to each of the internal decisions made by the hidden computational units of the distributed learning system, recognizing that those decisions are responsible for the overall outcomes in the first place.

- *Back Propagation Algorithm (BPA)*



Back Propagation Algorithm: Signal Flow



Notations



-
- Error signal
 - $e_j(n) = d_j(n) - y_j(n)$
 - MSE or the instantaneous error energy
 - $\mathcal{E}(n) = 1/2 \sum_{j=C} e_j^2(n)$
 - *C: neurons in the output layer*



BPA

- Induced local field $v_j(n)$

$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n)$$

- Function signal $y_j(n)$

$$y_j(n) = \phi_j(v_j(n))$$

- According to the gradient descent algorithm

- Weight correction $\Delta w_{ji}(n) \propto \partial \varepsilon(n) / \partial w_{ji}(n)$

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

BPA

- $\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} * \frac{\partial e_j(n)}{\partial y_j(n)} * \frac{\partial y_j(n)}{\partial v_j(n)}$
* $\frac{\partial v_j(n)}{\partial w_{ji}(n)}$
- $\frac{\partial \xi(n)}{\partial e_j(n)} =$
 - $e_j(n)$
- $\frac{\partial e_j(n)}{\partial y_j(n)} =$
 - -1
- $\frac{\partial y_j(n)}{\partial v_j(n)} =$
 - $\varphi'_j(v_j(n))$
- $\frac{\partial v_j(n)}{\partial w_{ji}(n)} =$
 - $y_i(n)$
- $\frac{\partial \xi(n)}{\partial w_{ji}(n)} = ?$

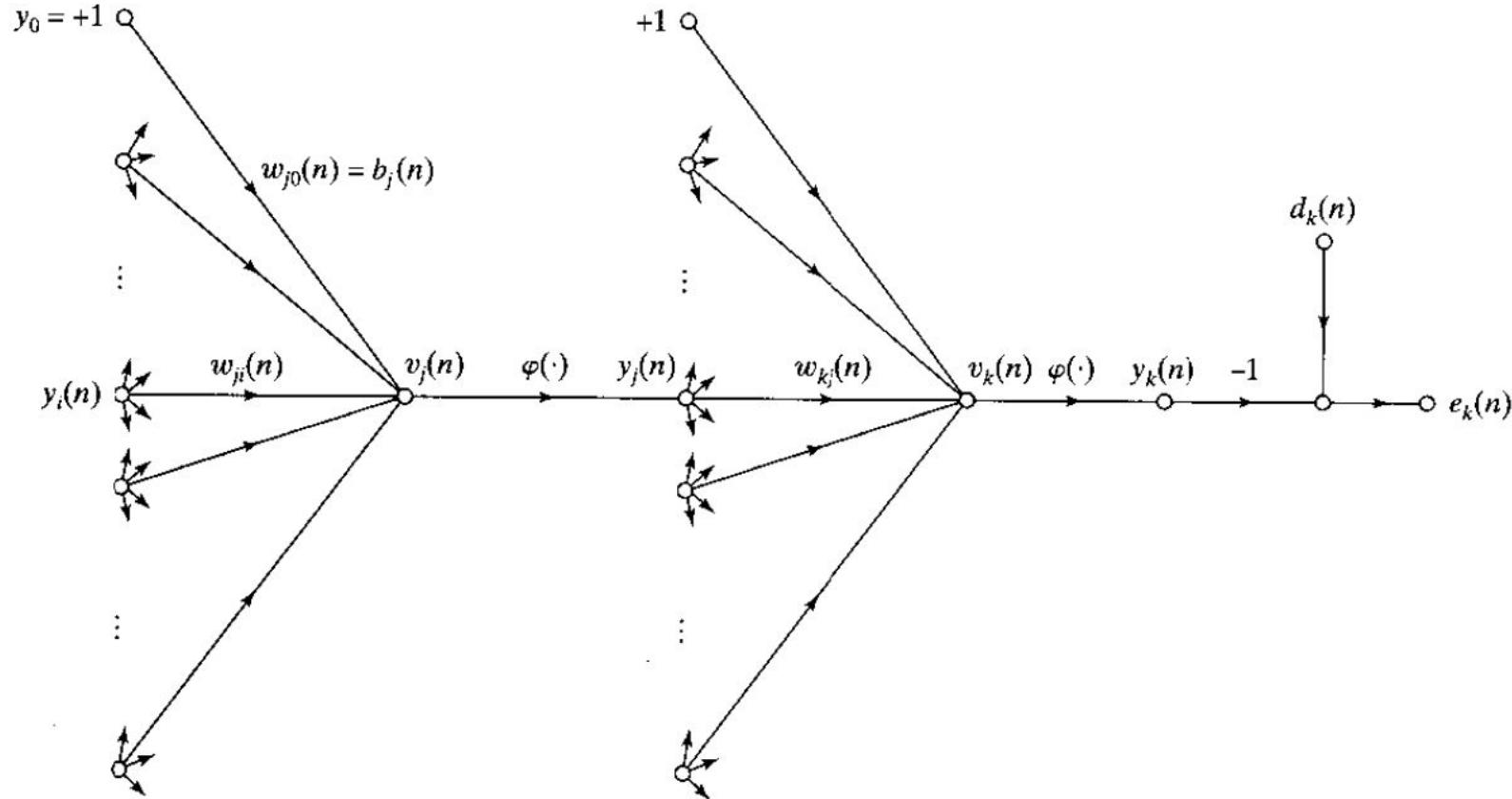
BPA

- $\frac{\partial \xi(n)}{\partial w_{ji}(n)} = \frac{\partial \xi(n)}{\partial e_j(n)} * \frac{\partial e_j(n)}{\partial y_j(n)} * \frac{\partial y_j(n)}{\partial v_j(n)}$
 $* \frac{\partial v_j(n)}{\partial w_{ji}(n)}$
- $\frac{\partial \xi(n)}{\partial e_j(n)} =$
 - $e_j(n)$
- $\frac{\partial e_j(n)}{\partial y_j(n)} =$
 - -1
- $\frac{\partial y_j(n)}{\partial v_j(n)} =$
 - $\varphi'_j(v_j(n))$
- $\frac{\partial v_j(n)}{\partial w_{ji}(n)} =$
 - $y_i(n)$
- $\frac{\partial \xi(n)}{\partial w_{ji}(n)} = -e_j(n) \varphi'_j(v_j(n)) y_i(n)$

BPA

- Weight correction $\Delta w_{ji}(n)$ is defined by the delta rule
 - $\Delta w_{ji}(n) = -\eta \partial \varepsilon(n) / \partial w_{ji}(n)$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$
 - Local gradient $\delta_j(n) = -\partial \varepsilon(n) / \partial v_j(n)$
 $= -\partial \varepsilon(n) / \partial e_j(n) * \partial e_j(n) / \partial y_i(n) * \partial y_i(n) / \partial v_j(n)$
 - $\delta_j(n) = e_j(n) \varphi'_j(v_j(n))$
- Case 1: Neuron j is an output node
 - $e_j(n) = d_j(n) - y_j(n)$
 - $\delta_j(n) = (d_j(n) - y_j(n)) \varphi'_j(v_j(n))$
- Case 1: Neuron j is a hidden node
 - $\delta_j(n)$ is recursively determined in terms of the local gradients of all neurons to which neuron j is directly connected

Back Propagation Algorithm: Signal Flow



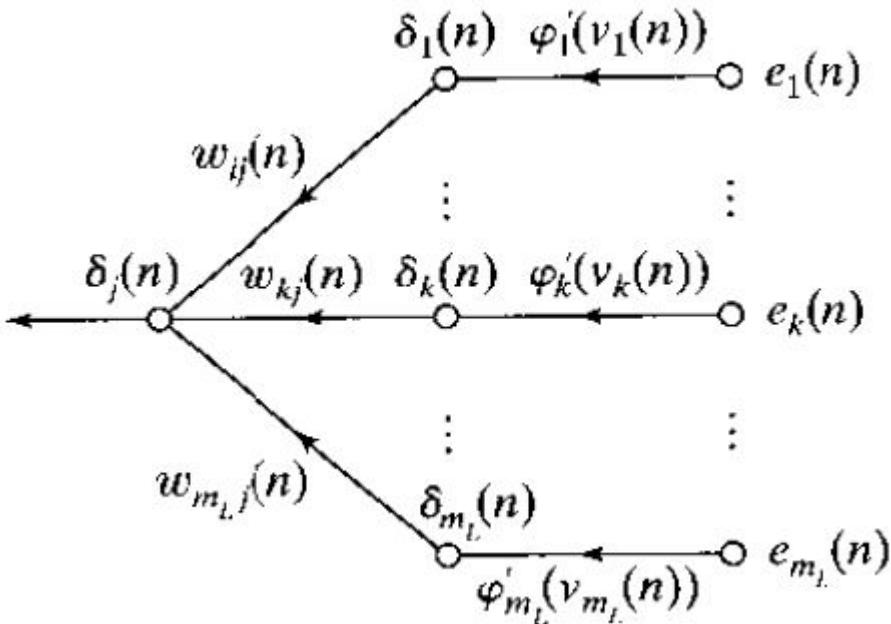
Local gradient: Hidden Neuron

- Local gradient $\delta_j(n) = -\partial \varepsilon(n)/\partial v_j(n)$
= $-\partial \varepsilon(n)/\partial y_j(n) * \partial y_j(n)/\partial v_j(n)$
= $-\partial \varepsilon(n)/\partial y_j(n)\varphi'_j(v_j(n))$
- $\varepsilon(n) = 1/2 \sum_{k=C} e_k^2(n)$
- $\partial \varepsilon(n)/\partial y_j(n) = \sum_k e_k(n) \partial e_k(n)/\partial y_j(n)$
- Applying chain rule of partial derivation
 - $\partial \varepsilon(n)/\partial y_j(n) = \sum_k e_k(n) \partial e_k(n)/\partial v_k(n) * \partial v_k(n)/\partial y_j(n)$
- From figure:
 - $e_k(n) = d_k(n) - y_k(n)$
 - $e_k(n) = d_k(n) - \varphi_k(v_k(n))$
- $\partial e_k(n)/\partial v_k(n) =$
 - $-\varphi'_k(v_k(n))$

Local gradient: Hidden Neuron

- $v_k(n) = \sum_m w_{kj}(n)y_j(n)$
 - m : total number of inputs applied to neuron k
- $\partial v_k(n)/\partial y_j(n) = w_{kj}(n)$
- $\partial \varepsilon(n)/\partial y_j(n) = \sum_k e_k(n) \partial e_k(n)/\partial y_j(n)$
 - $- \sum_k e_k(n) \varphi'(v_k(n)) w_{kj}(n)$
 - $- \sum_k \delta_k(n) w_{kj}(n)$
- Finally, $\boldsymbol{\delta}_j(n) = \boldsymbol{\varphi}'_j(v_j(n)) \sum_k \boldsymbol{\delta}_k(n) \boldsymbol{w}_{kj}(n)$

Error Propagation



- Delta rule
 - $\Delta w_{ji}(n) = \eta \delta_j y_i$
- Output Neuron
 - $\delta_j(n) = (d_j(n) - y_j(n))\varphi'_j(v_j(n))$
- Hidden Neuron
 - $\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$

Two Passes of Computation: Round Trip



- Forward Pass:
 - $y_j(n) = \varphi_j(v_j(n))$
 - $v_k(n) = \sum_m w_{ji}(n)y_i(n)$
 - m : total number of inputs applied to neuron j
 - $w_{ji}(n)$: Synaptic weight connecting neuron i to j
- Backward Pass:
 - $\Delta w_{ji}(n) = \eta \delta_j y_i$
 - Output Neuron:
 - $\delta_j(n) = (d_j(n) - y_j(n))\varphi'_j(v_j(n))$
 - Hidden Neuron:
 - $\delta_j(n) = \varphi'_j(v_j(n))\sum_k \delta_k(n)w_{kj}(n)$

Activation Function-I



- Sigmoidal Function

$$\phi_j(v_j(n)) = \frac{1}{1 + \exp(-av_j(n))}, a > 0; -\infty < v_j(n) < \infty$$

- Differentiation:

$$\phi'_j(v_j(n)) = \frac{a \exp(-av_j(n))}{[1 + \exp(-av_j(n))]^2}$$

- $y_j(n) = \phi_j(v_j(n))$

$$\phi'_j(v_j(n)) = ay_j(n)[1 - y_j(n)]$$

Local Gradient: Sigmoidal Function



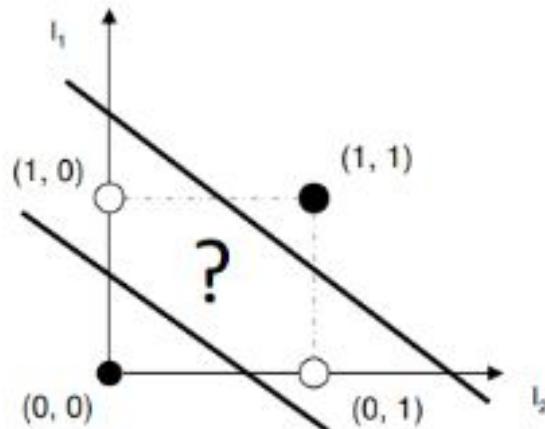
- $\Delta w_{ji}(n) = \eta \delta_j y_i$
- *Output Neuron:*
 - $\delta_j(n) = (d_j(n) - y_j(n))\varphi'_j(v_j(n))$
 - $y_j(n) = o_j(n)$
 - $\delta_j(n) = a[d_j(n) - o_j(n)]o_j(n)[1-o_j(n)]$
- *Hidden Neuron:*
 - $\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$
 - $\delta_j(n) = ?$
- *Maximum and Minimum of $\varphi'_j(v_j(n))$?*
 - *Synaptic weights are changed most where the function signals are in their mid-range.*

Example:XOR Problem

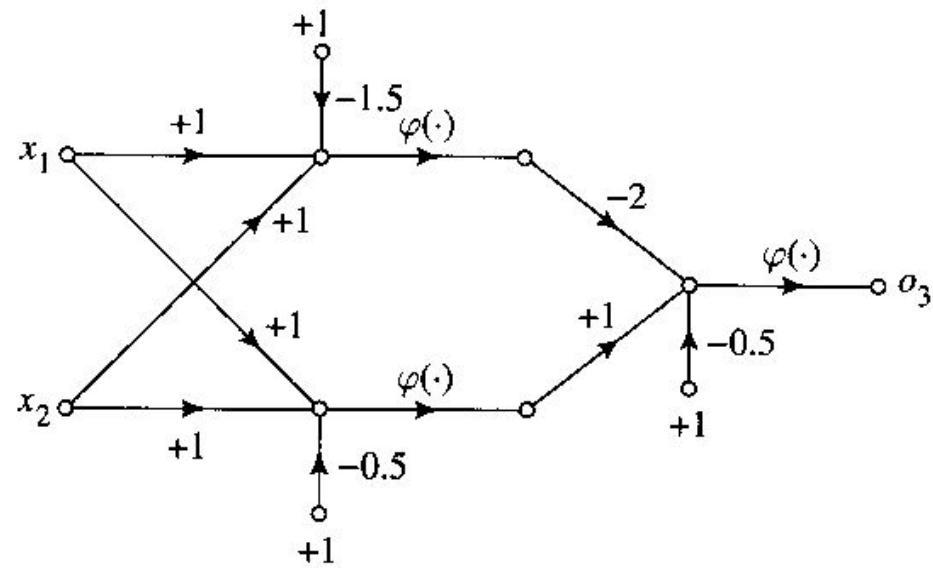
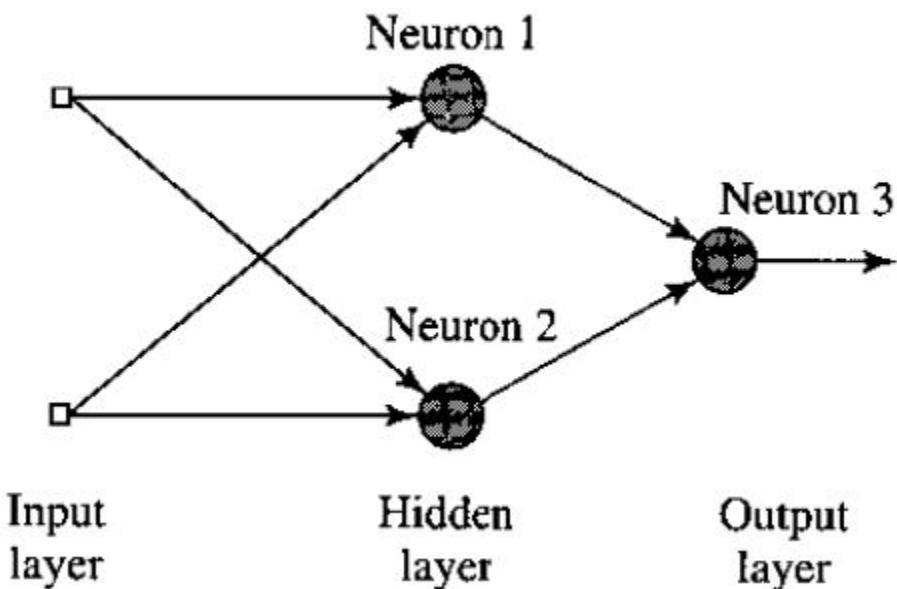


TABLE 6.2 XOR Problem

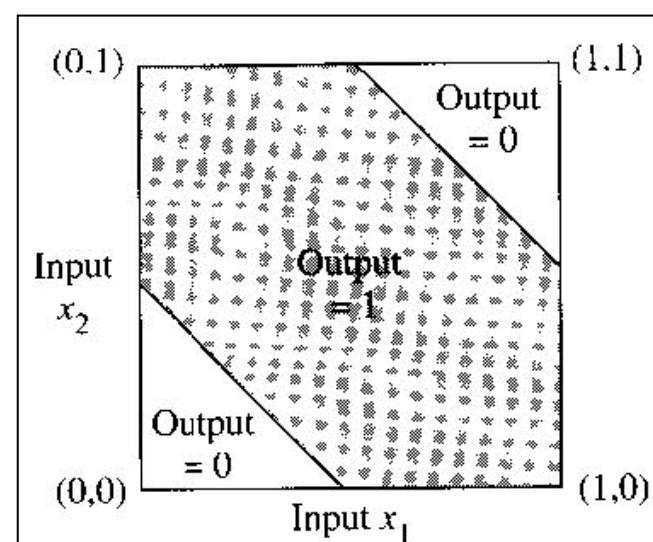
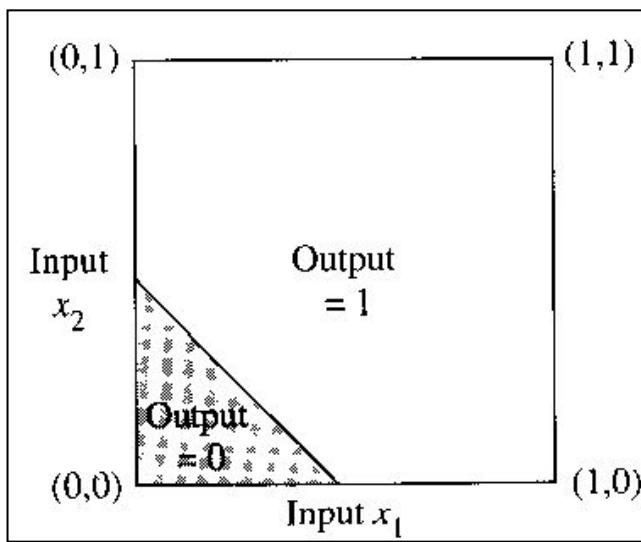
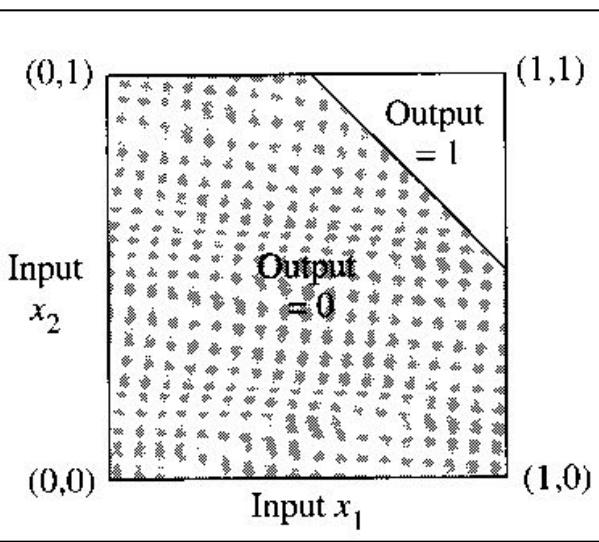
Input vector x	Desired response d
(-1, -1)	-1
(-1, +1)	+1
(+1, -1)	+1
(+1, +1)	-1



Example:XOR Problem



XOR Problem: Decision Boundary



Practical Considerations



-
- How are the weights initialised?
 - How is the learning rate chosen?
 - How many hidden layers and how many neurons?
 - Which activation function ?
 - How to preprocess the data ?
 - How many examples in the training data set?



Initial Weights



- We generally start off all the weights with small random values.
 - Gaussian distribution around zero with standard deviation σ .
- If synaptic weights are assigned large initial values neurons are driven into saturation.
- If synaptic weights are assigned small initial values algorithms operate around the origin.

Learning Rate



- Values between 0.1 and 0.9 have been used in many applications.
- Smaller learning rate -> slower rate of learning
 - Smaller changes in the synaptic weights
 - Smoother trajectory in the weight space
- Larger learning rate -> speed up the rate of learning
 - Larger changes in the synaptic weights
 - Unstable (oscillatory) network
- There is no necessity to keep the learning rate fixed throughout the learning process: $\eta(t) = \eta(1)/t$
- Generalized Delta Rule: $\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_j(n)$
 - α : Momentum constant

Number of Layers and Neurons



- The number of layers and of neurons depend on the specific task. In practice this issue is solved by trial and error.
- Two types of adaptive algorithms can be used:
 - start from a large network and successively remove some neurons and links until network performance degrades.
 - begin with a small network and introduce new neurons until performance is satisfactory.



Maximizing Information Content

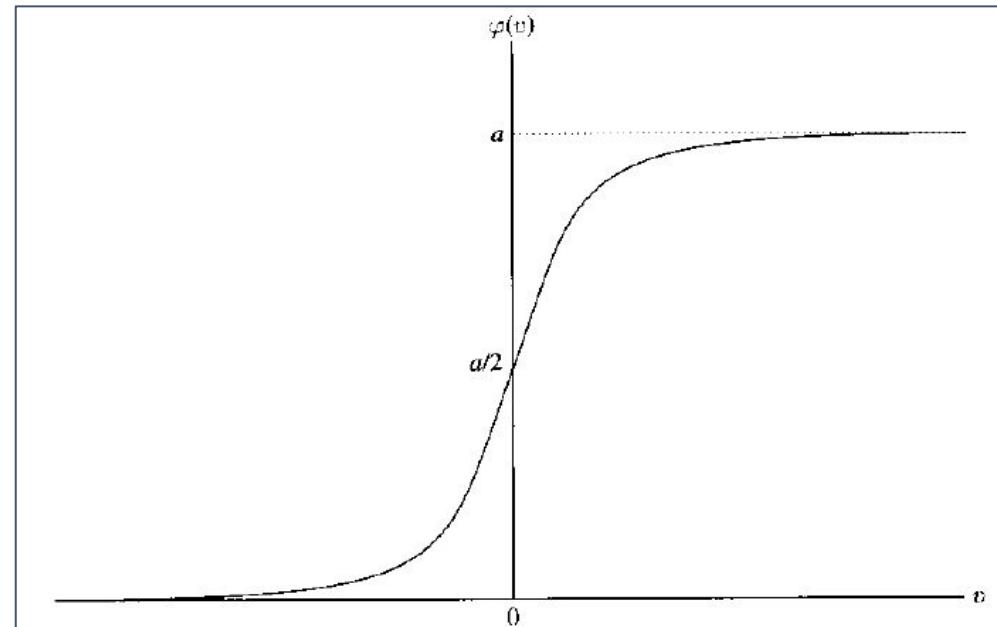
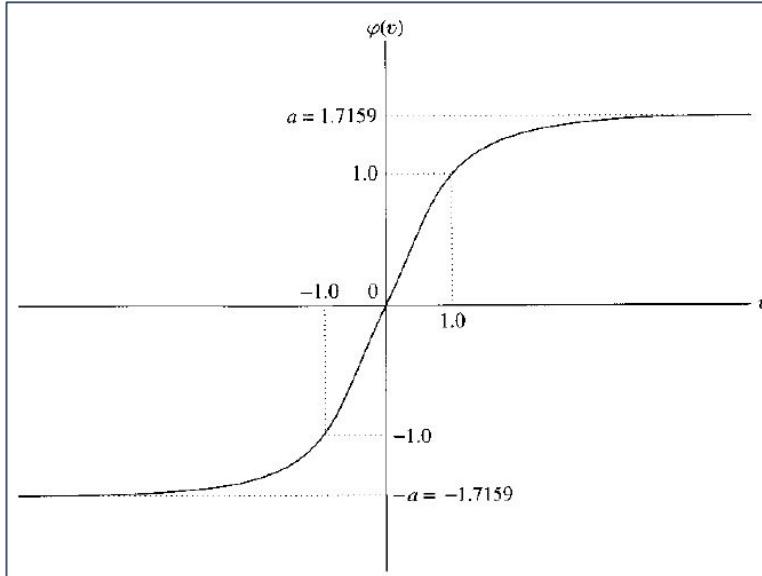


-
- Maximization of information content: every training example presented to the backpropagation algorithm must maximize the information content.
 - The use of an example that results in the largest training error.
 - The use of an example that is radically different from all those previously used.

Activation Function



- Network learns faster with antisymmetric functions when compared to non symmetric functions.
 - $\varphi(-v) = -\varphi(v)$
 - $\varphi(v) = \operatorname{atanh}(bv)$



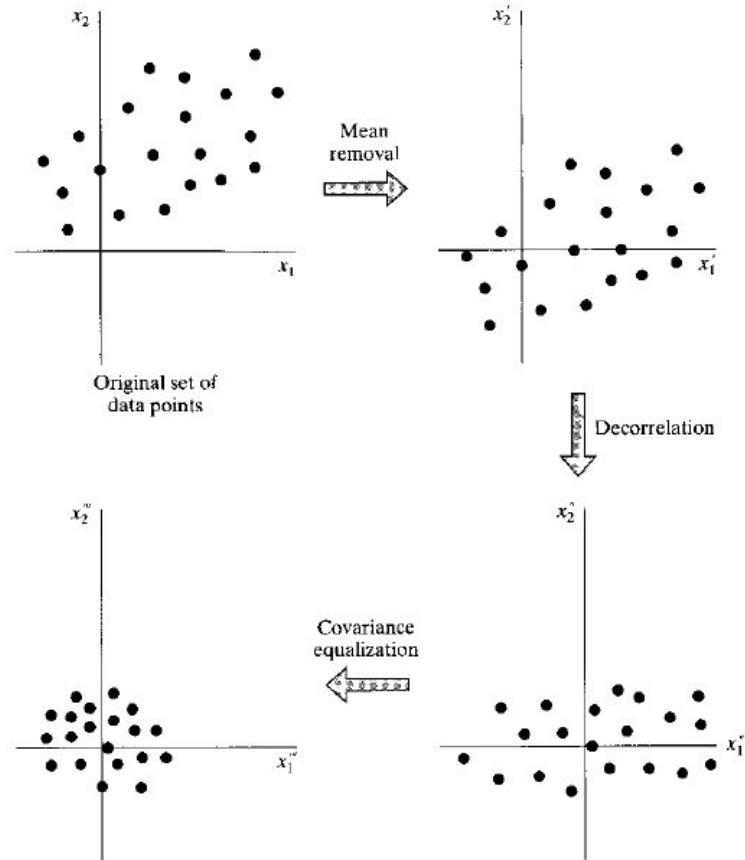
Target Values



- Target values: target values must be chosen within the range of the activation function.
- For the antisymmetric activation function it is necessary to design ε
 - For $a+$:
 - $d_j = a - \varepsilon$
 - For $-a$:
 - $d_j = -a + \varepsilon$
- If $a=1.7159$ we can set $\varepsilon = 0.7159$ then $d=\pm 1$
- We must use targets of $+1$ and -1 rather than 0 and 1 .

Normalization

- Each input variable should be preprocessed so that its mean value, averaged over the entire training sample is zero.
- The input variables in the training data should be uncorrelated.
- The decorrelated input variables should be scaled so that their covariances are approximately equal.



References



1. Chapter 6, 6.3-6.5, Neural Networks: A Comprehensive Foundation (2nd Edition) 2nd Edition by Simon Haykin
2. <https://www.youtube.com/watch?v=nz3NYD73H6E&list=PL3EA65335EAC29EE8&index=19>
3. https://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/lecture-notes/MIT15_097S12_lec13.pdf
4. https://people.eecs.berkeley.edu/~jordan/kernels/o521813972c03_p47-84.pdf
5. <http://www.cs.bham.ac.uk/~jxb/NN/l6.pdf>
6. <https://web.stanford.edu/group/pdplab/originalpdphandbook/Chapter%205.pdf>
7. <http://francky.me/aifaq/FAQ-comp.ai.neural-net.pdf>
8. <http://www.iro.umontreal.ca/~pift6266/Ao6/refs/YannNipsTutorial.pdf>



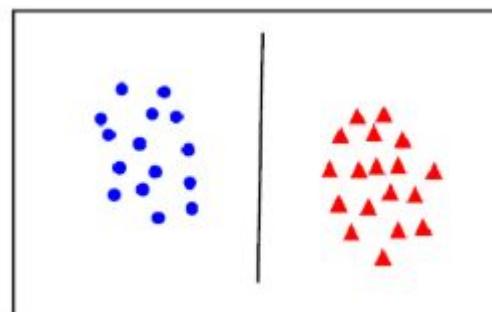
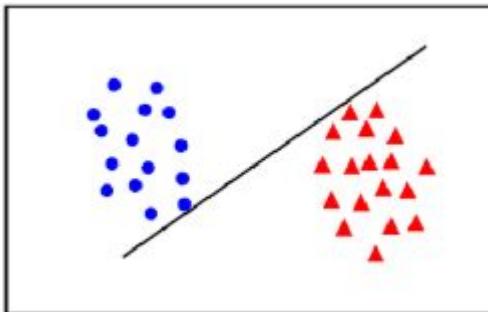
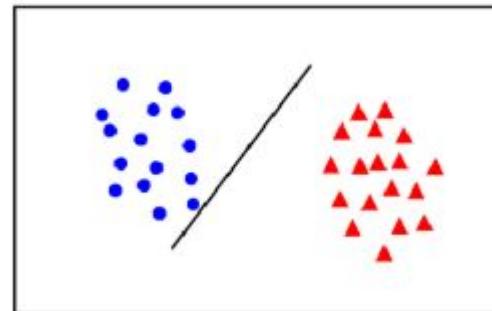
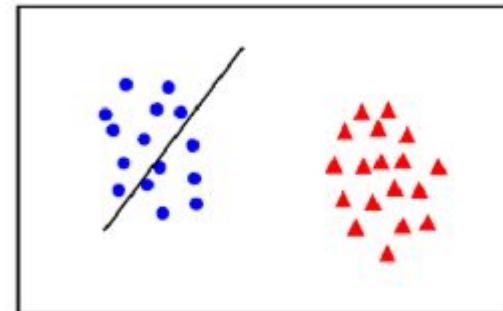
Support Vector Machines



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

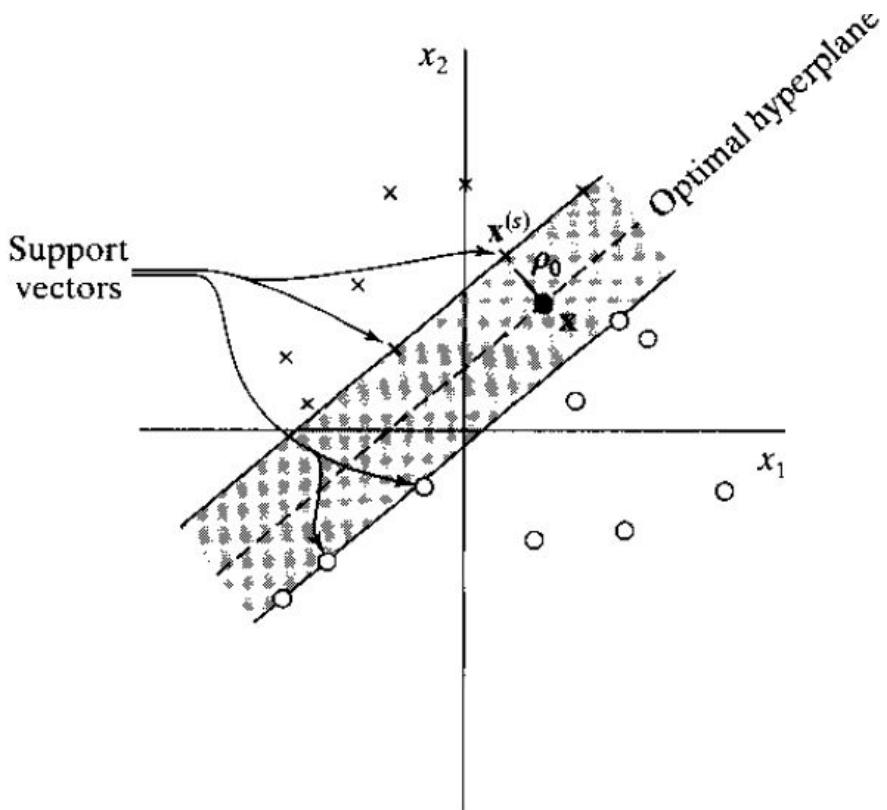


Optimum Separation Hyperplane



- Optimum separation hyperplane (OSH) is the linear classifier with the maximum margin for a given finite set of learning patterns.
- Better generalization!

Optimum Separation Hyperplane

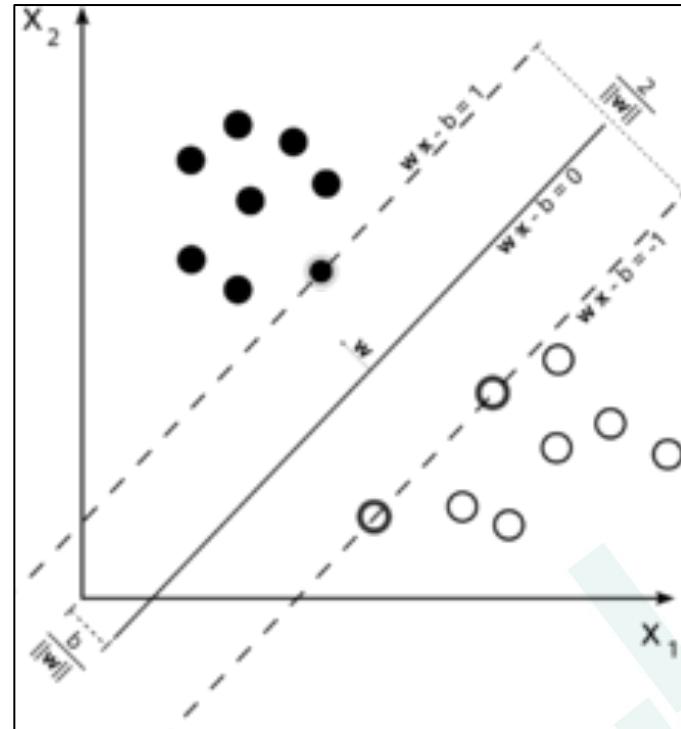


- *Margin of separation:* distance to the closest example
- For the optimal hyperplane
 - distance to the closest negative example = distance to the closest positive example
- The goal of SVM is to find the particular hyperplane for which the *margin of separation* is maximized.

Support Vectors



- *Support vectors* are the samples closest to the separating hyperplane
 - They are the most difficult patterns to classify.
- Optimal separation hyper-plane is completely defined by *support vectors*



Optimal Hyperplane: Problem Formulation

- Training Set: $D = \{(x_i, d_i); i = 1, 2, \dots, n\}$
- Linearly Separable:

- The Decision Boundary [Hyperplane]

$$\sum_{i=1}^m w_i x_i + b = W^T x + b = 0$$

- Correct Classification

$$w^T x_i + b \geq 0; \forall y_i = +1$$

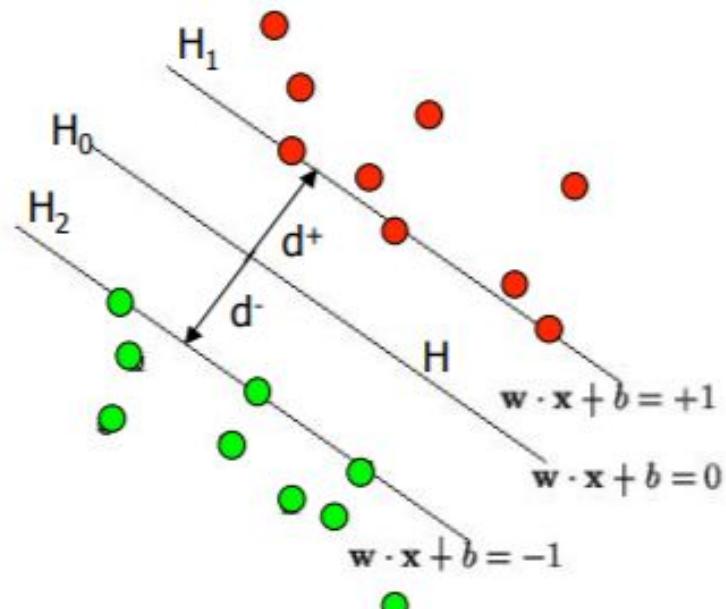
$$w^T x_i + b < 0; \forall y_i = -1$$

- Infinitely many hyperplanes exist
 - Which is the optimal?

Margin of Separation



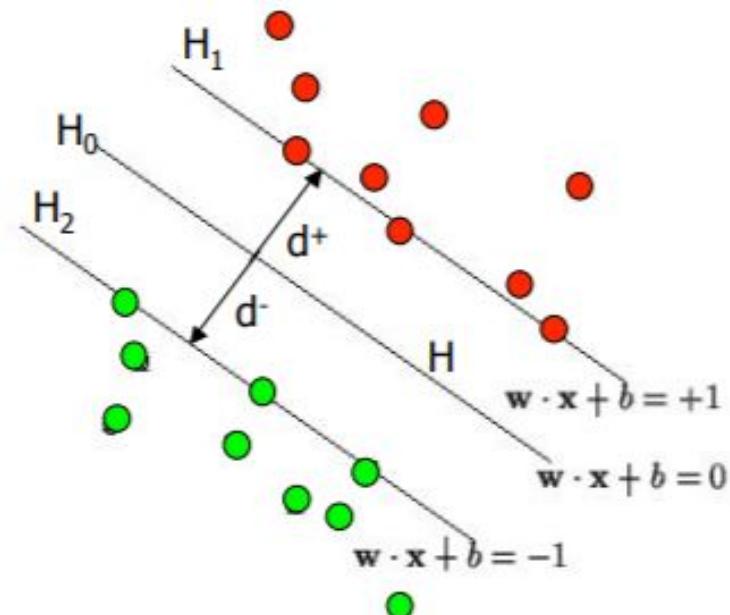
- NO training patterns exist between the two hyperplanes:
 - $H_1: w \cdot x + b = 1, y = 1$
 - $H_2: w \cdot x + b = -1, y = -1$
- The points on the planes H_1 and H_2 are the Support Vectors
- d^+ = the shortest distance to the closest positive point
- d^- = the shortest distance to the closest negative point
- The margin m of a separating hyperplane is $(d^+) + (d^-)$



Maximizing the margin



- We want a classifier (linear separator) with as big a margin as possible.
- Distance from a point (x_o, y_o) to a Line $Ax + By + c = 0$ is
 - $|Ax_o + By_o + c|/\sqrt{A^2 + B^2}$
- The distance between H_0 and H_1
 - $|w \cdot x + b|/\|w\| = 1/\|w\|$
- The total distance m between H_1 and H_2 :
 - $2/\|w\|$



Quadratic Programming Problem



- When $\| w \| = 1$ then $m=2$
- When $\| w \| = 2$ then $m=1$
- When $\| w \| = 4$ then $m=1/2$
- The bigger the norm is, the smaller the margin become.
- *Maximize $2/\| w \|$*
 - *Minimize $\| w \| / 2$*
 - $= \text{Minimize } 1/2 \| w \|^2$
- ***Minimize f: $1/2\|w\|^2$ s.t. g: $y_i[w \cdot x_i + b] \geq 1$***
- This is a constrained optimization problem
 - It can be solved by the Lagrangian multiplier method
 - Because it is quadratic, the surface is a paraboloid, with just a single global minimum

SVM: Constrained Optimization Problem



- Given the training sample $\{(x_i, y_i)\}_{i=1}^n$, find the optimum values of the weight vector \mathbf{w} and bias \mathbf{b} such that they satisfy the constraints
 - $y_i^*(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \forall i=1,2,\dots,n$
 - Equality is true for support vector points and greater than condition holds true for non-support vector points.

and the weight vector \mathbf{w} minimizes the cost function:

- $\Phi(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2 = \frac{1}{2} \mathbf{w}^T \mathbf{w}$
- Constraints are linear
- Cost function is convex

Constrained Optimization



- *Lagrangian function: Constrained optimization can be solved through unconstrained optimization*
 - $L(x,y,a) = f(x,y) - ag(x,y)$
 - a : Lagrange multipliers
- *Solution: $\nabla L(x,y,a) = 0$*
 - $\partial L(x,y,a)/\partial x = 0$
 - $\partial L(x,y,a)/\partial y = 0$
 - $\partial L(x,y,a)/\partial a = 0$



Lagrange Multiplier: Primal Form



- $J(w, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^n \alpha_i [y_i^* (\mathbf{w}^T \mathbf{x}_i + b)]$
 - Inequality constraints \rightarrow equality constraints
 - $J(w, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^n \alpha_i [y_i^* (\mathbf{w}^T \mathbf{x}_i + b) - 1]$
- Karush-Kuhn-Tucker Condition
 - Multipliers that can assume non-zero values ($\alpha > 0$), must satisfy following conditions:
 - $\alpha_i [y_i^* (\mathbf{w}^T \mathbf{x}_i + b) - 1] = 0$

Lagrange Multiplier



- *Solution:* $\nabla L(x,y,a) = 0$
 - *Conditions of optimality*
 - $\partial J(\mathbf{w}, b, a)/\partial \mathbf{w} = 0$
 - $\partial J(\mathbf{w}, b, a)/\partial b = 0$
- $J(w, b, a) = 1/2 \mathbf{w}^T \mathbf{w} - \sum_{i=1}^n \alpha_i [y_i^* (\mathbf{w}^T \mathbf{x}_i + b) - 1]$
 - $1/2 \mathbf{w} \cdot \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{w} \cdot \mathbf{x}_i - \sum_{i=1}^n \alpha_i y_i b + \sum_{i=1}^n \alpha_i$
- $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \dots \dots \text{(A)}$
- $\sum_{i=1}^n \alpha_i y_i = 0 \dots \dots \text{(B)}$

Duality theorem (Bertsekas, 1995)



- If the primal problem has an optimal solution, the dual problem also has an optimal solution, and the corresponding optimal values are equal.
 - $\Phi(\mathbf{w}_o) = J(\mathbf{w}_o, b_o, \alpha_o) = \min J(w, b, \alpha)$
- Applying (B) and then (A) to Lagrangian equation:
 - $J(w, b, \alpha) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w} - \sum_{i=1}^n \alpha_i y_i \mathbf{w} \cdot \mathbf{x}_i + \sum_{i=1}^n \alpha_i$
 - $J(w, b, \alpha) = \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^n \alpha_i$
 - $Q(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$

Dual Problem



- Given the training sample $\{(x_i, y_i)\}_{i=1}^n$, find the Lagrange multipliers that maximize the objective function
 - $$Q(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j, \text{ S.T.}$$
 - $\alpha_i \geq 0, \forall i = 1, 2, \dots, n$
 - $\sum_{i=1}^n \alpha_i y_i = 0$
- Primal vs Dual
 - The dual problem is cast entirely in terms of the training data
- Objective: To find the lagrangian multipliers which maximizes the $Q(\alpha)$
 - *Some of the lagrangian multipliers will become zero*
 - *Some of the lagrangian multipliers will have high value*

Interpretation



-
- *Some of the lagrangian multipliers will have high value*
 - Corresponding input training sample is a support vector
 - *Some of the lagrangian multipliers will become zero*
 - Corresponding input training sample is not a support vector
 - *Some of the lagrangian multipliers might have very high value*
 - Corresponding input training sample is an outlier



Optimal weight and bias: Decision boundary

- Having determined the optimum Lagrangian multipliers, the optimum weight vector may be computed
 - $\mathbf{w}_o = \sum_{i=1}^{n_s} \alpha_{o,i} y_i \mathbf{x}_i$
 - n_s is the number of support vectors for which the Lagrange multipliers are all non-zero
- Having obtained w_o , the bias b_o may be computed
 - $b_o = 1 - w_o^T \mathbf{x}^{(s)}, \forall y^{(s)} = 1$
 - $1 - \sum_{i=1}^{n_s} \alpha_{o,i} y_i \mathbf{x}_i \mathbf{x}^{(s)}$
- For a new sample z , calculate $\mathbf{w}_o z + b$
 - $\sum_{i=1}^{n_s} \alpha_{o,i} y_i \mathbf{x}_i z_j + (1 - \sum_{i=1}^{n_s} \alpha_{o,i} y_i \mathbf{x}_i \mathbf{x}^{(s)})$
- If the sign is positive, the sample z will belong the positive class, else the sample z will belong to the negative class.

Breakout Room Activity



i	x_i	y_i	α_i	i	x_i	y_i	α_i
1	(4,2.9)	1	0.414	6	(1.9,1.9)	-1	0
2	(4,4)	1	0	7	(3.5, 4)	1	0.018
3	(1,2.5)	-1	0	8	(0.5,1.5)	-1	0
4	(2.5,1)	-1	1.18	9	(2,2.1)	-1	0.414
5	(4.9,2.5)	1	0	10	(4.5,2.5)	1	0

Consider the training data samples and the corresponding Lagrange multipliers learned from them, as given in the following table.

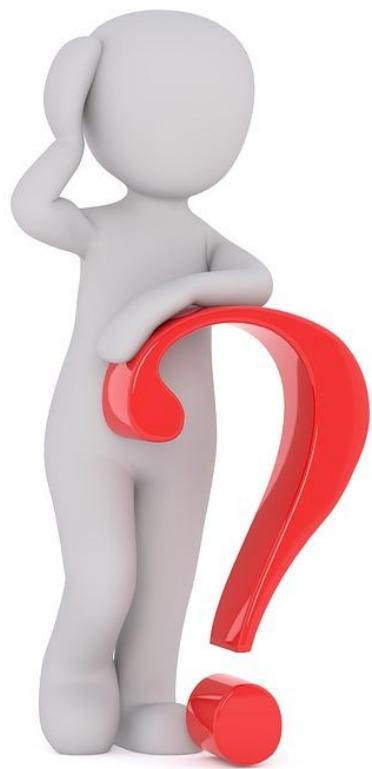
From the given table above, answer the following questions?

1. What is the b for the SVM?
2. Identify the support vectors.
3. Compute w and classify the point (3,3).

References



1. <https://www.svm-tutorial.com/2017/02/svms-overview-support-vector-machines/>
2. https://www.syncfusion.com/ebooks/support_vector_machines_succinctly/introduction
3. <https://www.youtube.com/watch?v=b-Su6aVh5yo>
4. Chapter 6, Neural Networks: A Comprehensive Foundation (2nd Edition) 2nd Edition by Simon Haykin



Supplement: Scaling of weight vectors



- Distance from a point (x_o, y_o) to a Line $Ax + By + c = 0$ is
 - $|Ax_o + By_o + c|/\sqrt{A^2 + B^2}$
- The distance between support vector point and H_o
 - $m = |w \cdot x_o + b|/\|w\|$
 - The geometric margin is clearly invariant to scaling of weight parameters because it is inherently normalized by the length of $\|w\|$
 - This means that we can impose any scaling constraint we wish on $\|w\|$ without affecting the geometric margin.
- $|w \cdot x_o + b| = m^* \|w\|$
- By applying a proper scaling to the weights, the factor $m^* \|w\|$ can always be made = 1
 - $|w \cdot x + b| = 1$ for support vector points

Support Vector Machine for Non-Linearly Separable Patterns



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



Non-linearly Separable Data



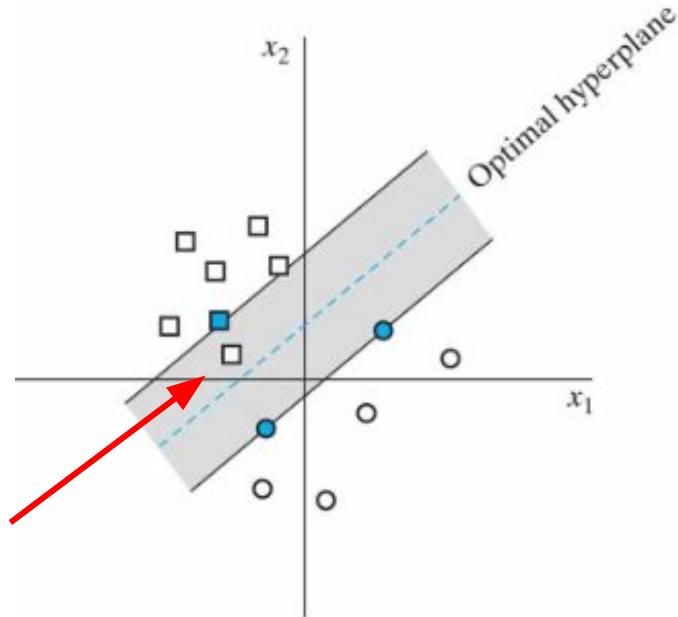
-
- Linear SVM with Soft Margin
 - Non-linear SVM
 - Kernel SVM



Non-Separable Patterns



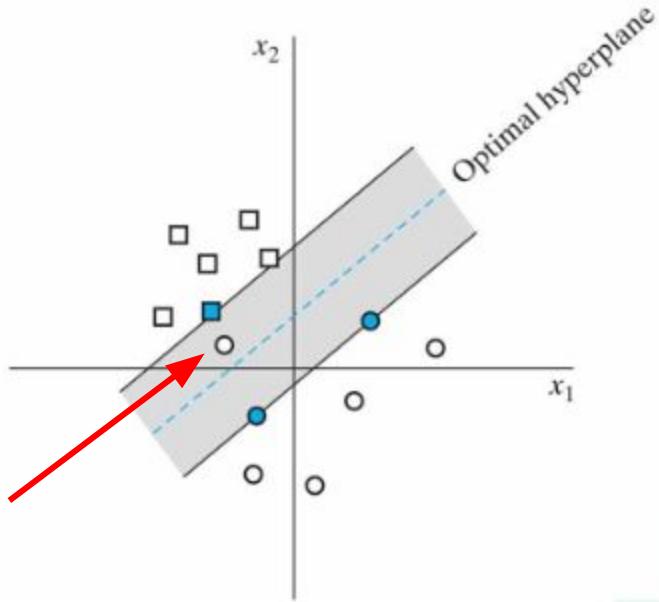
- $y_i^* (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \forall i=1,2,\dots,N$
- *Violation:*
 - $\{(x_i, y_i)\}$ falls on the right side of the decision surface



Non-Separable Patterns



- $d_i^* (\mathbf{w}^T \mathbf{x}_i + b) >= 1, \forall i=1,2, \dots N$
- *Violation:*
 - $\{(x_i, y_i)\}$ falls on the wrong side of the decision surface



Slack Variables



- Let $\{\varepsilon_i\}_{i=1}^N \geq 0$
 - $y_i^*(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \varepsilon_i, \forall i=1,2,\dots,N$
- ε_i are known as slack variables:
 - Deviation of a data point from the ideal condition of pattern separability
 - $0 \leq \varepsilon_i \leq 1$: Data point falls on the right side of the decision surface
 - $\varepsilon_i > 1$: Data point falls on the wrong side of the decision surface
- Support vectors are those particular data points that satisfy the equation precisely even if $\varepsilon_i > 0$

Optimal Hyperplane



- The cost function
 - $\Phi(\mathbf{w}, \varepsilon) = \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \varepsilon_i$
 - C: Regularization parameter that controls the tradeoff between the complexity of the machine and the number of nonseparable patterns.
- High value of C -> High confidence in the quality of the training data
- Small value of C -> Less emphasis on the training data
- C has to be selected by the user *experimentally*.

Primal Problem



- Given the training sample $\{(x_i, y_i)\}_{i=1}^N$, find the optimum values of the weight vector w and bias b such that they satisfy the constraints
 - $y_i^*(w^T x_i + b) \geq 1 - \varepsilon_i, \forall i=1,2,\dots,N$
 - $\varepsilon_i \geq 0, \forall i$
- and such that the weight vector w and the slack variables ε_i minimize the cost functional
 - $\Phi(w, \varepsilon) = 1/2 w^T w + C \sum_{i=1}^N \varepsilon_i$
where C is user-specified positive parameter.
- Exercise: Apply the Lagrangian multipliers to get the dual*

Dual Problem

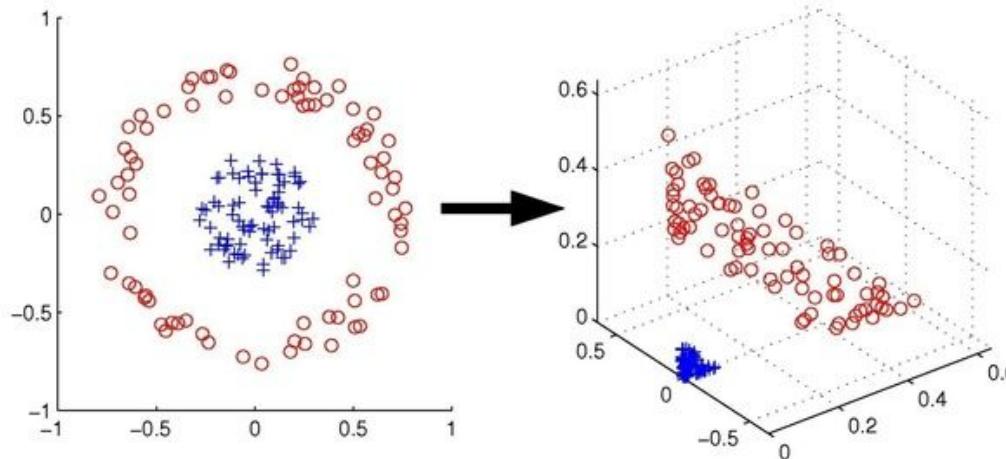


- Given the training sample $\{(x_i, y_i)\}_{i=1}^N$, find the Lagrange multipliers α_i that maximize the objective function
 - $Q(\alpha) = \sum_{i=1}^N \alpha_i - 1/2 \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$, S.T.
 - $\sum_{i=1}^N \alpha_i y_i = 0$
 - $0 \leq \alpha_i \leq C, \forall i = 1, 2, \dots, N$
- where C is user-specified positive parameter
- Neither the slack variables nor their own Langarne multipliers appear in the dual problem!
- Separable vs Non Separable
 - $\alpha_i \geq 0$
 - $0 \leq \alpha_i \leq C, \forall i = 1, 2, \dots, N$

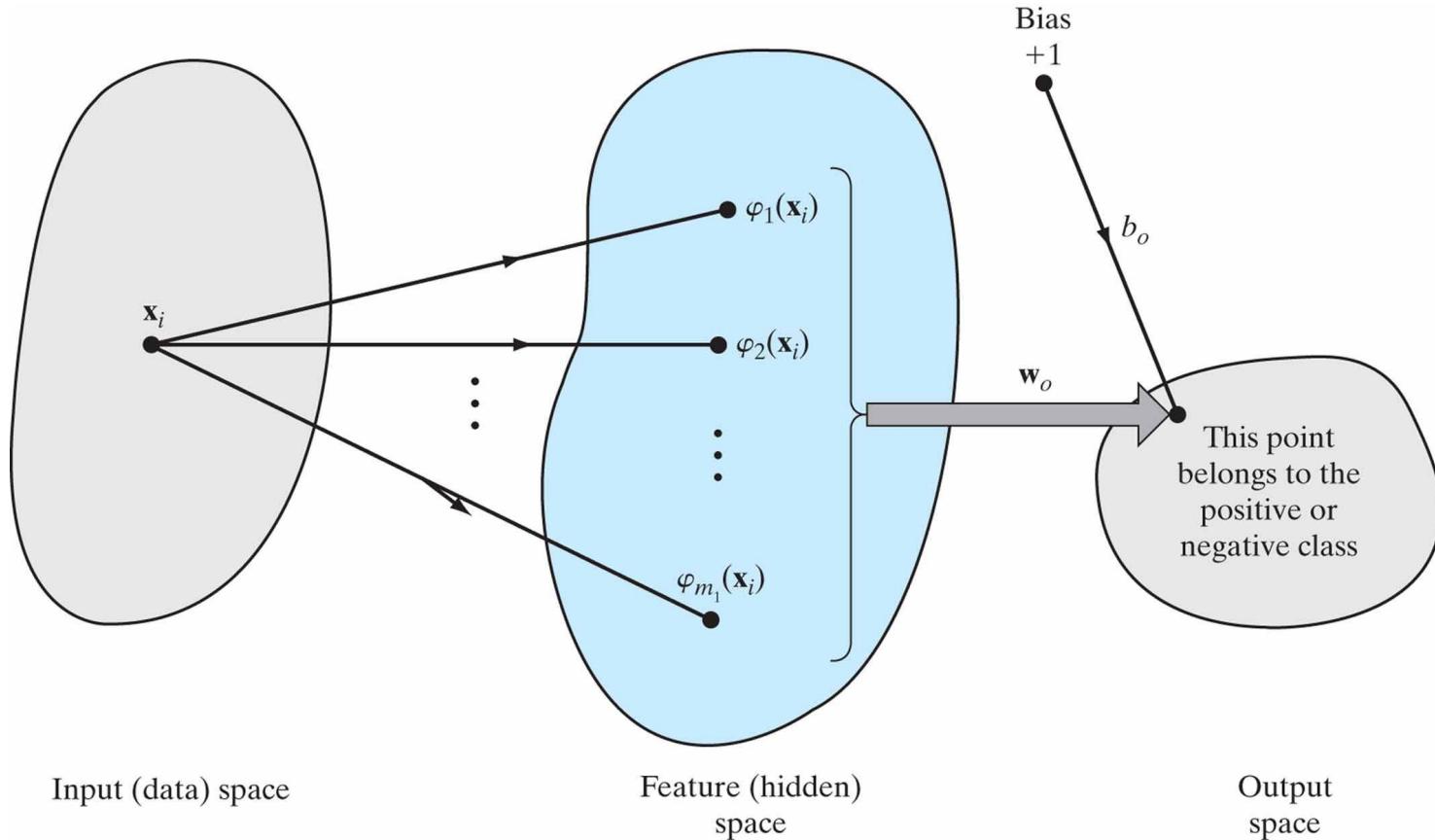
Cover's Theorem:(Cover, 1965)



- “A complex pattern-classification problem cast in high-dimensional space nonlinearly is more likely to be linearly separable than in a low dimensional space”
 - Transformation is non-linear
 - Dimensionality of the feature space is high enough



Kernel SVM: Non-linear Mapping



Kernel SVM: Optimal Hyperplane in Feature Space



- Similar to the idea of the non separable patterns
 - Separating hyperplane is defined as a linear function of vectors drawn from the feature space rather than the original input space.
- Non-linear Transformation

$$\Phi(x) = [\Phi_0(x), \Phi_1(x), \dots, \Phi_m(x)]$$

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y_i x_i \quad \mathbf{w} = \sum_{i=1}^N \alpha_i y_i \Phi(x_i)$$

$$\mathbf{w}^T \Phi(x) = 0 \quad \sum_{i=1}^N \alpha_i y_i \Phi^T(x_i) \Phi(x) = 0$$

The Inner Product Kernel



- For a given set of training samples (in a lower-dimensional feature space) and a transformation into a higher-dimensional space, there exists a function (The Kernel Function) which can compute the dot product in the higher-dimensional space without explicitly transforming the vectors into the higher-dimensional space first.

$$K(\mathbf{x}, \mathbf{x}_i) = \Phi(\mathbf{x})^T \Phi(\mathbf{x}_i)$$
$$= \sum_{j=0}^m \Phi_j(x)^T \Phi_j(x_i), \forall i = 1, 2, \dots, N$$
$$K(\mathbf{x}, \mathbf{x}_i) = K(\mathbf{x}_i, \mathbf{x}), \forall i$$

The Kernel Trick



- Optimal Hyperplane

$$\sum_{i=1}^N \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) = 0$$

- Specifying the kernel is *sufficient* for the pattern classification in the output space; we need never explicitly compute the weight vector w_o .
- The optimal hyperplane consists of a finite number of terms that is equal to the number of support vector patterns.

Optimum Design of a SVM



- Given the training sample $\{(x_i, y_i)\}_{i=1}^N$, find the Lagrange multipliers α_i that maximize the objective function
 - $Q(\alpha) = \sum_{i=1}^N \alpha_i - 1/2 \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j)$ S.T.
 - $\sum_{i=1}^N \alpha_i y_i = 0$
 - $0 \leq \alpha_i \leq C, \forall i = 1, 2, \dots, N$
- where C is user-specified positive parameter
- $K(x_i, x_j)$ can also be viewed as the ij -th element of a symmetric N -by- N matrix K :
 - $K = \{K(x_i, x_j)\}_{(i,j)=1}^N$
- Optimal weight $w_o = \sum_{i=1}^{N_s} \alpha_{o,i} y_i \phi(x_i)$
 - First component of w_o represents the bias b_o
- Feature Space dimensionality is determined by N_s .

Inner-Product Kernels

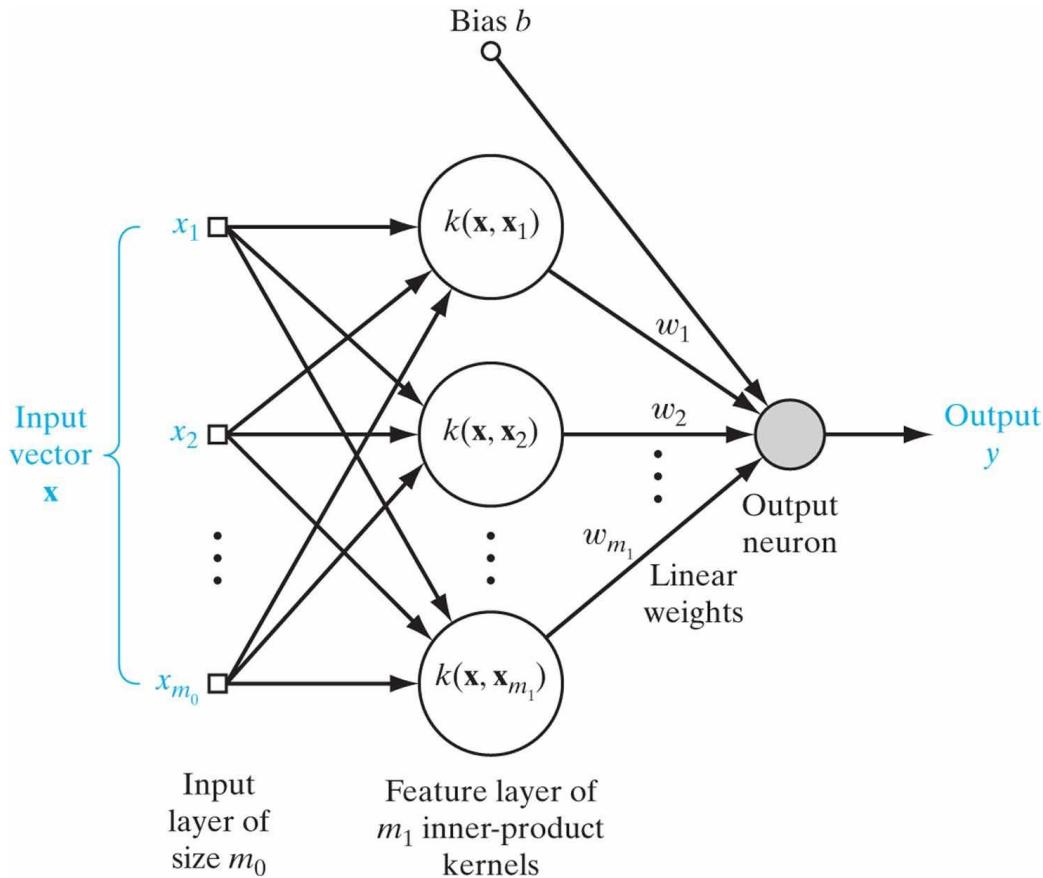


TABLE 6.1 Summary of Mercer Kernels

Type of support vector machine	Mercer kernel $k(\mathbf{x}, \mathbf{x}_i), i = 1, 2, \dots, N$	Comments
Polynomial learning machine	$(\mathbf{x}^T \mathbf{x}_i + 1)^p$	Power p is specified <i>a priori</i> by the user
Radial-basis-function network	$\exp\left(-\frac{1}{2\sigma^2} \ \mathbf{x} - \mathbf{x}_i\ ^2\right)$	The width σ^2 , common to all the kernels, is specified <i>a priori</i> by the user
Two-layer perceptron	$\tanh(\beta_0 \mathbf{x}^T \mathbf{x}_i + \beta_1)$	Merger's theorem is satisfied only for some values of β_0 and β_1

- *Exercise: Apply the Lagrangian multipliers to get the dual*
- *Exercise: Read Mercer Theorem*

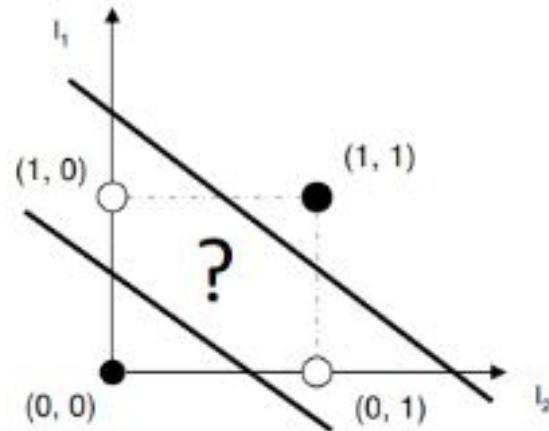
Architecture of SVM



XOR Problem: Kernel Trick

TABLE 6.2 XOR Problem

Input vector \mathbf{x}	Desired response d
(-1, -1)	-1
(-1, +1)	+1
(+1, -1)	+1
(+1, +1)	-1



- Given the following kernel:
 - $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}^T \mathbf{x})^2$
- Derive the optimum \mathbf{w} for a SVM using the above kernel

Example:XOR Problem



- Define a kernel:
 - $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{1} + \mathbf{x}_i^T \mathbf{x}_j)^2$
- Let $\mathbf{x} = [x_1, x_2]^T$, $\mathbf{x}_i = [x_{i1}, x_{i2}]^T$
- $K(\mathbf{x}_i, \mathbf{x}_j) = ?$
 - $1 + x_1^2 x_{i1}^2 + 2x_1 x_2 x_{i1} x_{i2} + x_2^2 x_{i2}^2 + 2x_1 x_{i1} + 2x_2 x_{i2}$
- $\phi(\mathbf{x}) = [1, x_1^2, \sqrt{2x_1 x_2}, x_2^2, \sqrt{2x_1}, \sqrt{2x_2}]$
- $\phi(\mathbf{x}_i) = [1, x_{i1}^2, \sqrt{2x_{i1} x_{i2}}, x_{i2}^2, \sqrt{2x_{i1}}, \sqrt{2x_{i2}}]$
- Gram

$$\mathbf{K} = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}$$

Example:XOR Problem



- Dual form Objective Function

- $$Q(\alpha) = \sum_{i=1}^N \alpha_i - 1/2 \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i \mathbf{x}_j),$$

$$\begin{aligned} Q(\alpha) = & \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2} (9\alpha_1^2 - 2\alpha_1\alpha_2 - 2\alpha_1\alpha_3 + 2\alpha_1\alpha_4 \\ & + 9\alpha_2^2 + 2\alpha_2\alpha_3 - 2\alpha_2\alpha_4 + 9\alpha_3^2 - 2\alpha_3\alpha_4 + 9\alpha_4^2) \end{aligned}$$

- *Conditions of optimality*

- $\partial Q(\alpha)/\partial \alpha = 0$

$$\begin{aligned} 9\alpha_1 - \alpha_2 - \alpha_3 + \alpha_4 &= 1 \\ -\alpha_1 + 9\alpha_2 + \alpha_3 - \alpha_4 &= 1 \\ -\alpha_1 + \alpha_2 + 9\alpha_3 - \alpha_4 &= 1 \\ \alpha_1 - \alpha_2 - \alpha_3 + 9\alpha_4 &= 1 \end{aligned}$$

Example:XOR Problem



- Dual form Objective Function Solution

$$\alpha_{o,1} = \alpha_{o,2} = \alpha_{o,3} = \alpha_{o,4} = \frac{1}{8}$$

- Optimum Weight

- $\mathbf{w}_o = \sum_{i=1}^{N_s} \alpha_{o,i} d_i \varphi(\mathbf{x}_i)$

$$\mathbf{w}_o = \frac{1}{8} [-\varphi(\mathbf{x}_1) + \varphi(\mathbf{x}_2) + \varphi(\mathbf{x}_3) - \varphi(\mathbf{x}_4)]$$

Example:XOR Problem



- Optimum Weight

$$\frac{1}{8} \left[- \begin{bmatrix} 1 \\ 1 \\ \sqrt{2} \\ 1 \\ -\sqrt{2} \\ -\sqrt{2} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -\sqrt{2} \\ 1 \\ -\sqrt{2} \\ \sqrt{2} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -\sqrt{2} \\ 1 \\ \sqrt{2} \\ -\sqrt{2} \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \\ \sqrt{2} \\ 1 \\ \sqrt{2} \\ \sqrt{2} \end{bmatrix} \right]$$

$$\begin{bmatrix} 0 \\ 0 \\ -1/\sqrt{2} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Example:XOR Problem



- Optimum Hyperplane

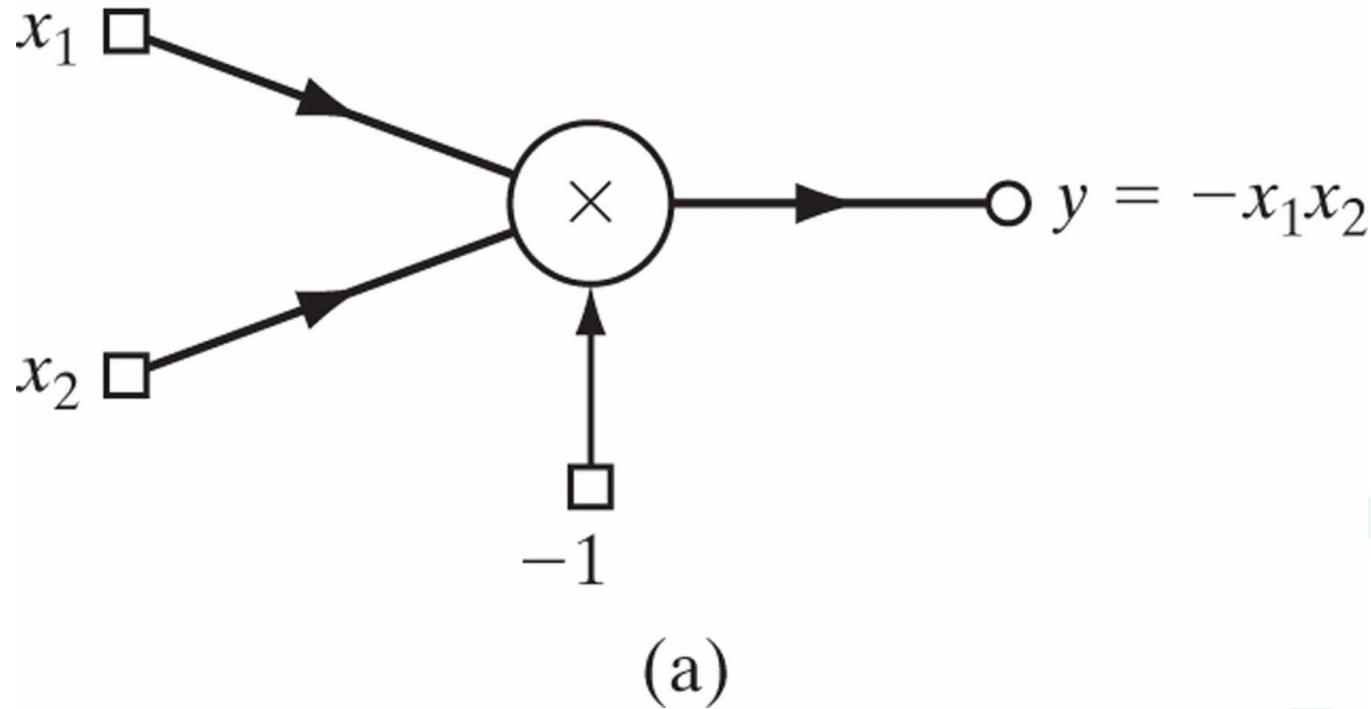
- $\mathbf{w}_o^T \phi(\mathbf{x}) = 0$ or

$$\sum_{i=1}^N \alpha_i d_i K(\mathbf{x}, \mathbf{x}_i) = 0$$

$$\left[0, 0, \frac{-1}{\sqrt{2}}, 0, 0, 0 \right] \begin{bmatrix} 1 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \end{bmatrix} = 0$$

$$-x_1x_2 = 0$$

Example:XOR Problem



References



1. <https://www.youtube.com/watch?v=SRVswRH5Q7E>
2. <http://cs229.stanford.edu/notes/cs229-notes3.pdf>
3. Chapter 6, Neural Networks: A Comprehensive Foundation (2nd Edition) 2nd Edition by Simon Haykin
4. Hastie, T., R. Tibshirani, and J. Friedman. The Elements of Statistical Learning, second edition. New York: Springer, 2008.
5. Christianini, N., and J. Shawe-Taylor. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge, UK: Cambridge University Press, 2000.
6. Fan, R.-E., P.-H. Chen, and C.-J. Lin. “Working set selection using second order information for training support vector machines.” Journal of Machine Learning Research, Vol 6, 2005, pp. 1889–1918.



Clustering



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



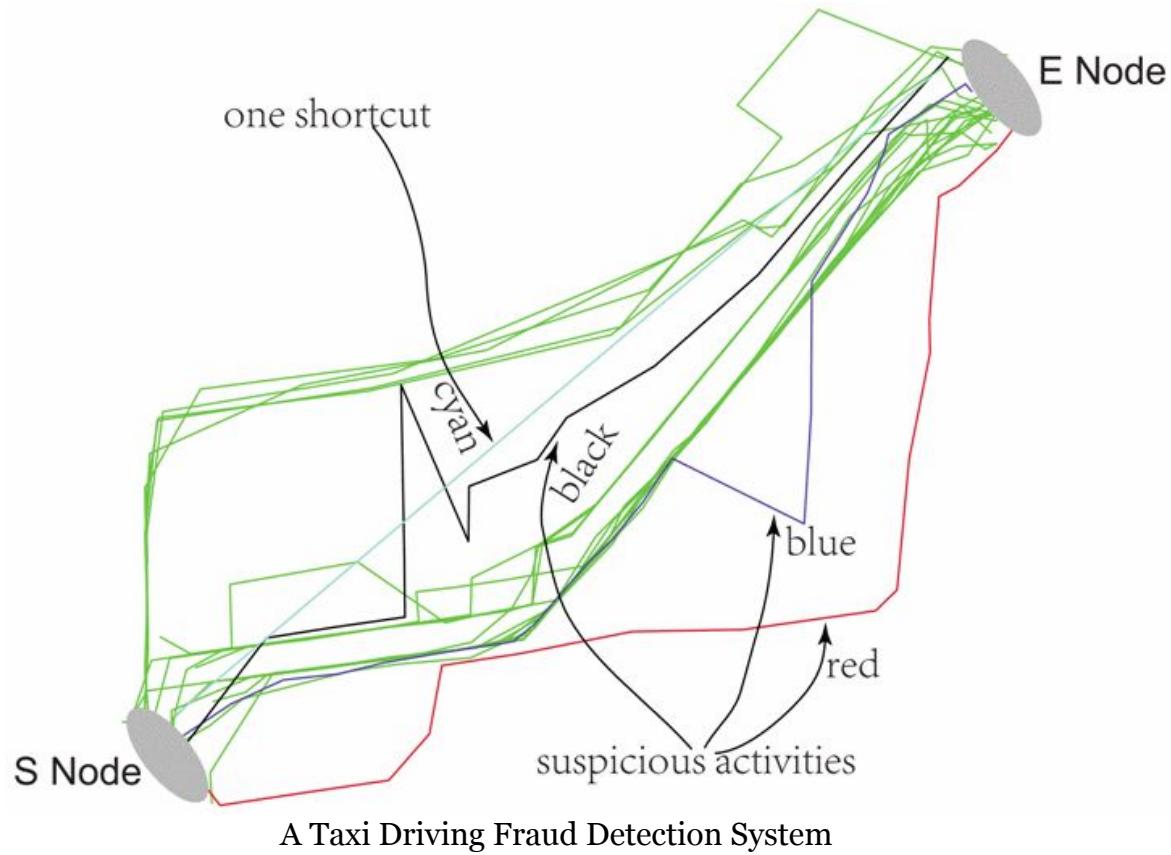
What is Clustering



- Organizing data into clusters such that there is
 - high intra-cluster similarity
 - low inter-cluster similarity
- Informally, finding natural groupings among objects



An interesting Use Case: Fraudulent Driving



Unsupervised Learning



- Clustering methods are unsupervised learning techniques
 - We do not have a teacher that provides examples with their labels
 - Requires data, but no labels
- Detect patterns e.g. in
 - Group emails or search results
 - Customer shopping patterns
 - Regions of images
- Useful
 - when don't know what you're looking for
 - Unavailability of the annotations

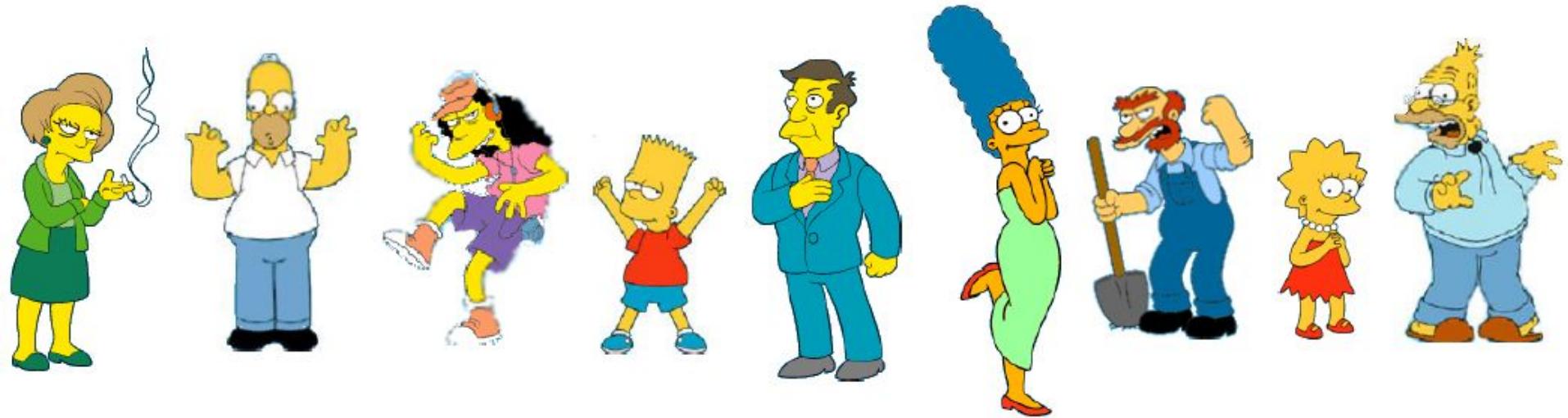
Why Clustering



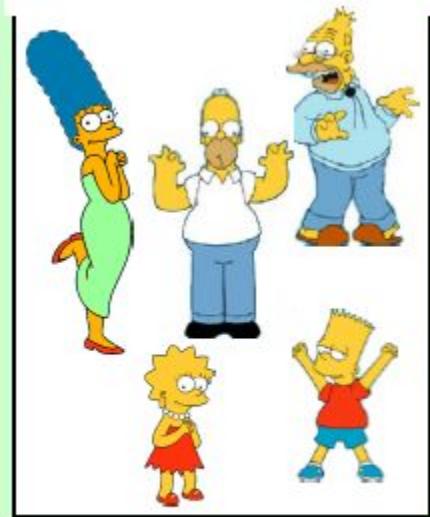
- Organizing data into clusters provides information about the internal structure of the data
 - Ex. detecting fraudulent driving
- Sometimes the partitioning is the goal
 - Image segmentation
 - Places which are contributing most to the pollution
- Knowledge discovery in data
 - Underlying rules, recurring patterns, topics, etc



Natural grouping : Clustering is subjective



Natural grouping : Clustering is subjective



Simpson's Family



School Employees



Females



Males

What is Similarity?



Distance Measures



- The distance (dissimilarity) between O_1 and O_2 is a real number denoted by $D(O_1, O_2)$.
- Euclidean distance
 - $d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$
- Correlation coefficient
 - $s(x, y) = \sum_i (x_i - \mu_x)(y_i - \mu_y)/\sigma_x \sigma_y$



Desirable Properties

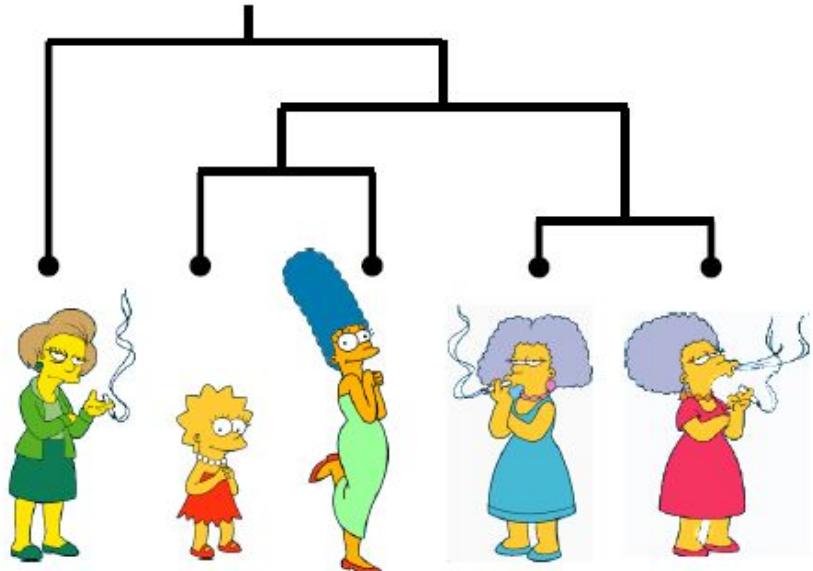


- Symmetric
 - $D(A, B) = D(B, A)$
 - Otherwise, we can say that A looks like B but B does not look like A
- Positivity and Self-similarity
 - $D(A, B) \geq 0$ and $D(A, B) = 0$, iff $A = B$
 - Otherwise, there will be different objects that we cannot tell apart
- Triangle Inequality
 - $D(A, B) + D(B, C) \geq D(A, C)$
 - Otherwise, we can say that A is like B, B is like C but A is not like C at all

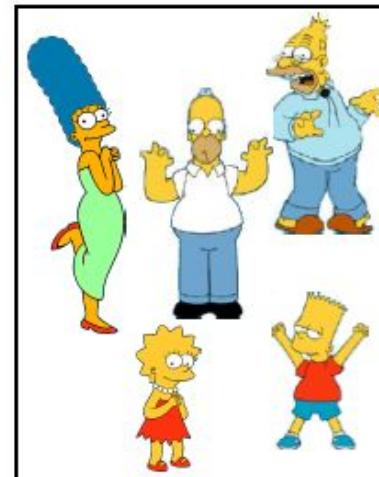
Clustering Types



Hierarchical



Partitional

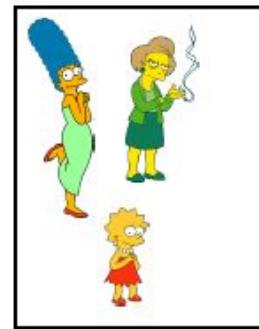
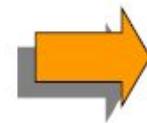
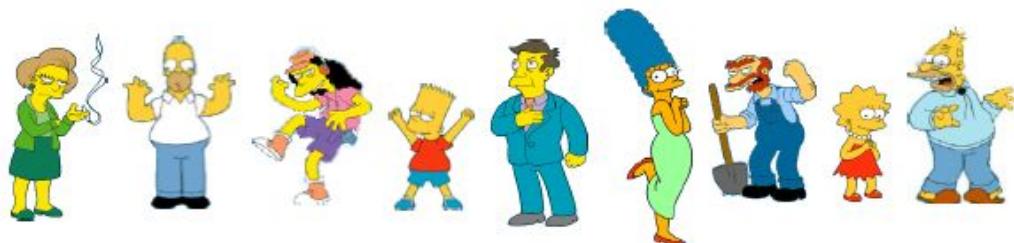


Partitional Clustering: K-Means Clustering

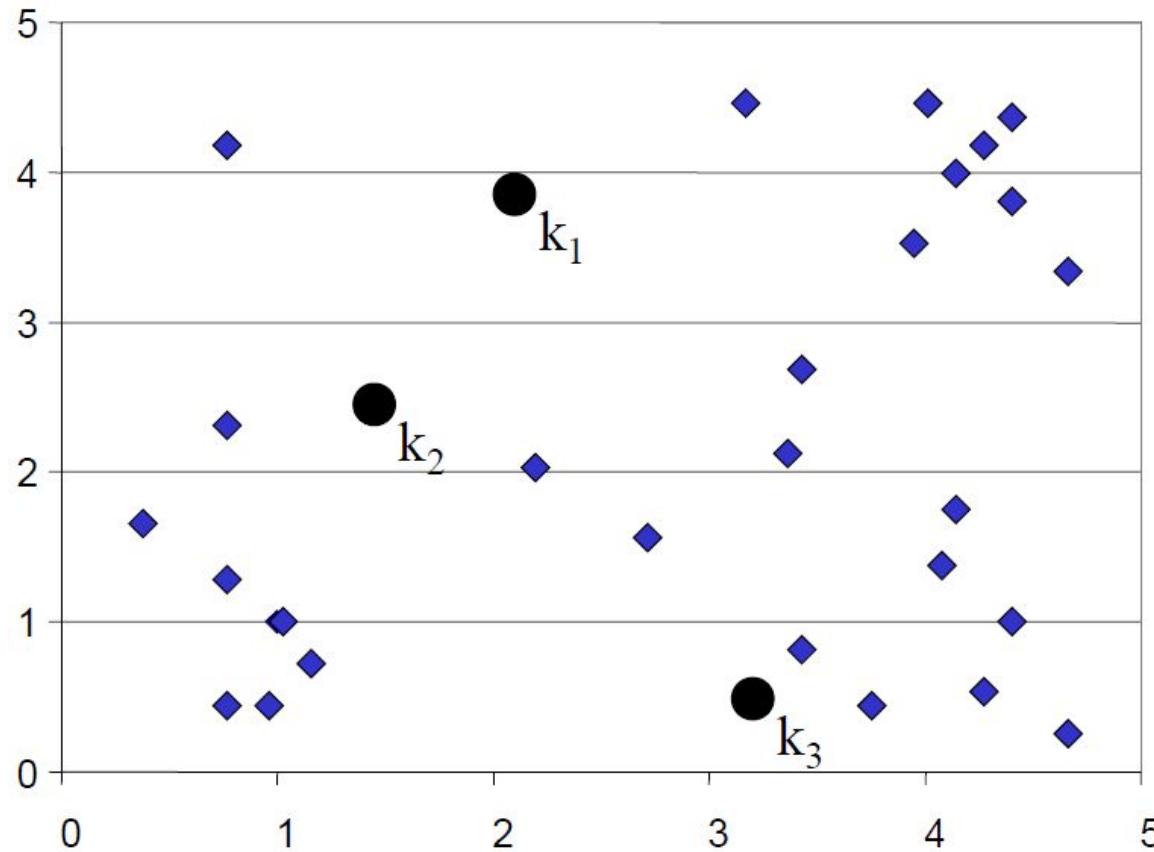


- Each cluster is associated with a centroid (center point)
- Each point is assigned to the cluster with the closest centroid
- Number of clusters, K , must be specified
- The objective is to minimize the sum of distances of the points to their respective centroid

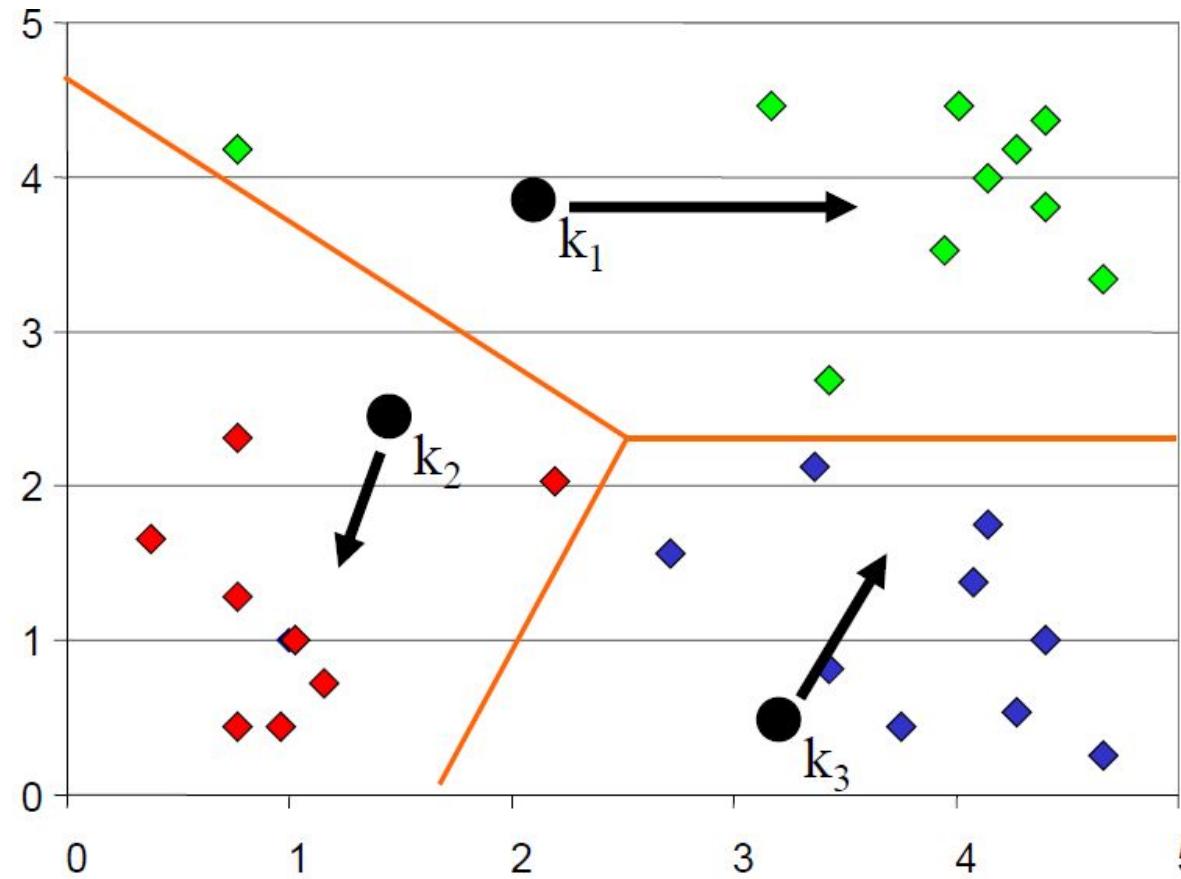
K-Means Clustering



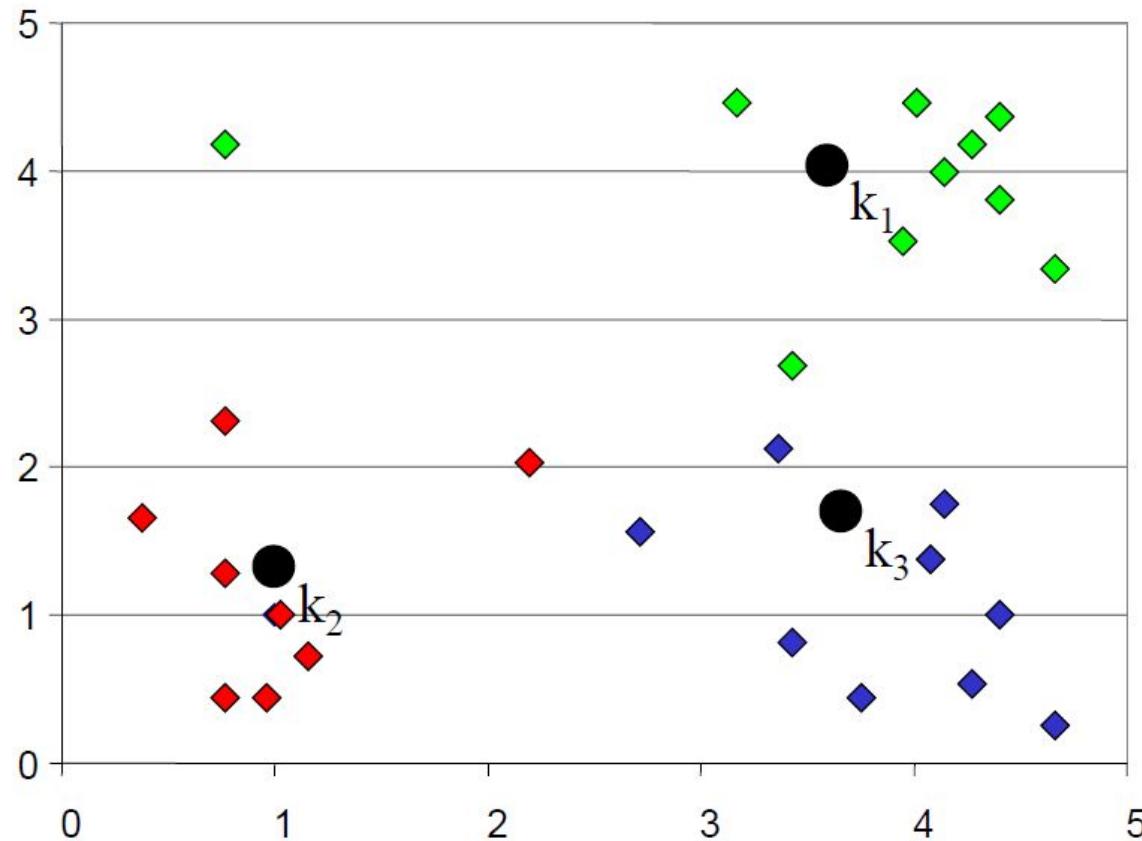
K-Means Clustering: Initialization



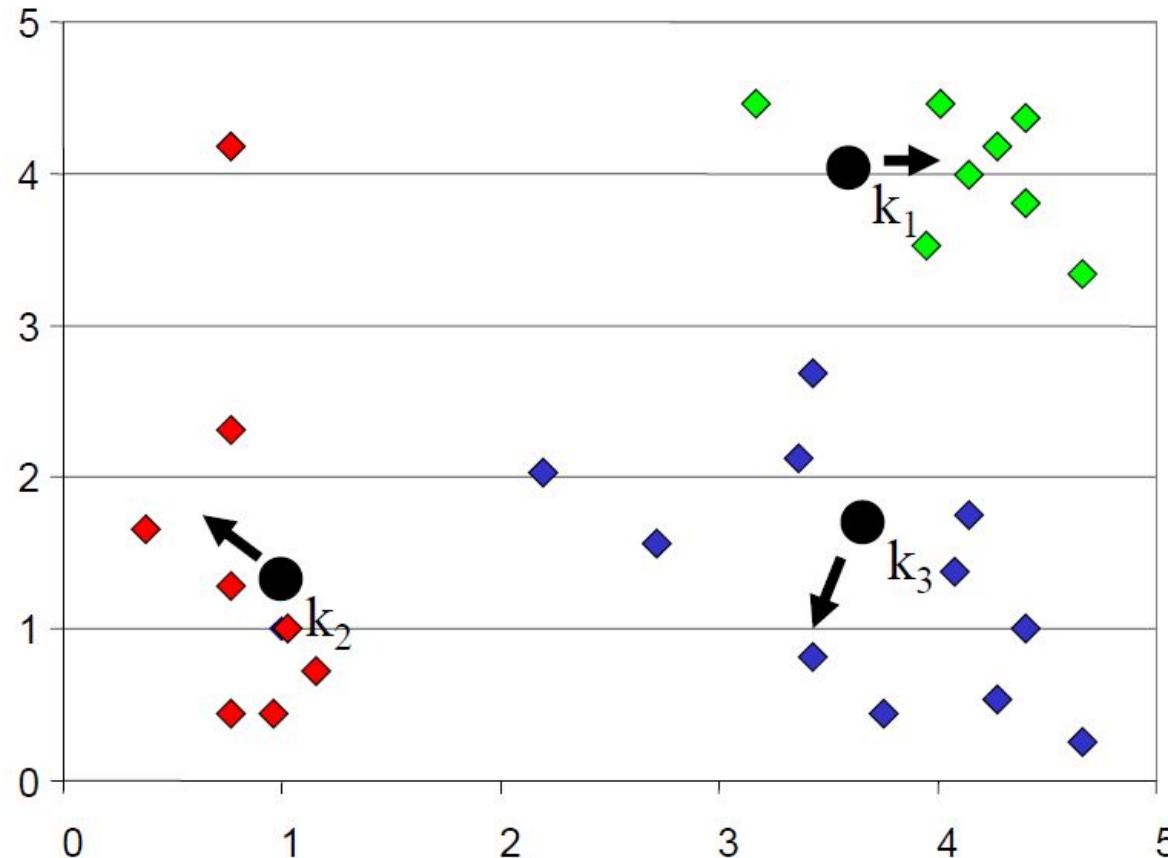
K-Means Clustering: Iteration-I



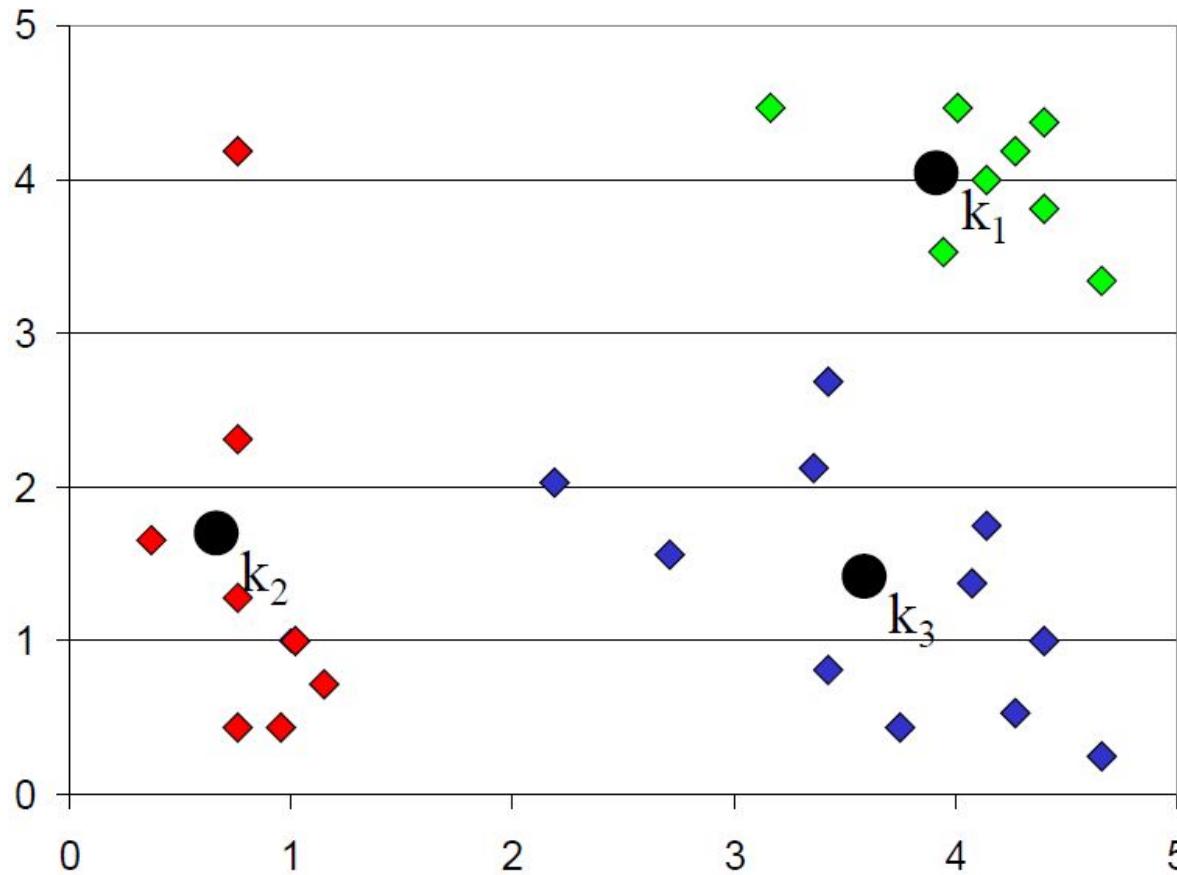
K-Means Clustering: Iteration-I



K-Means Clustering: Iteration-II



K-Means Clustering: Iteration-III



Algorithm: K-Means Clustering



-
- 0.1 Decide on a value for K , the number of clusters.
 - 0.2 Initialize the K cluster centers (randomly, if necessary).
 - 1. *Assignment:* Decide the class memberships of the n objects by assigning them to the nearest cluster center.

$$\min \sum_{i=1}^n |x_i - \mu_{x_i}|^2$$

- 2. *Re-estimate* the K cluster centers, by assuming the memberships found above are correct.

$$\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Repeat 3 and 4 until none of the n objects changed membership in the last iteration.

Convergence: K-Means Clustering



- Loss Function of the K-Means

$$L(C, \mu) = \min_{\mu} \min_C \sum_{i=1}^k \sum_{x \in C_i} |x - \mu_i|^2$$

- Fix μ optimize C

- Assign data points to closest cluster center

$$L(C^{t+1}, \mu^t) < L(C^t, \mu^t)$$

- Fix C optimize μ

- Change the cluster center to the average of its assigned points

$$L(C^{t+1}, \mu^{t+1}) \leq L(C^{t+1}, \mu^t)$$

- Loss function is guaranteed to decrease monotonically in each iteration in each steps until convergence.

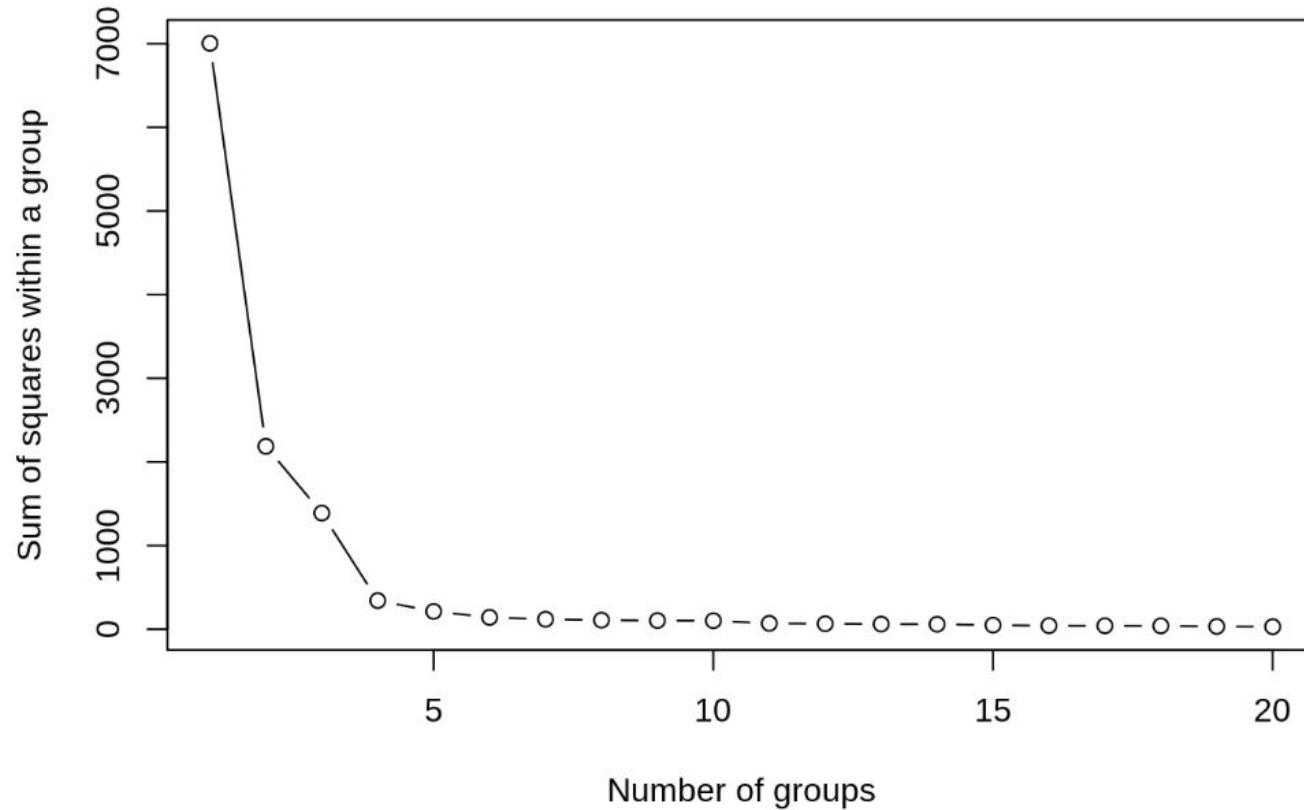
Properties: K-Means Clustering



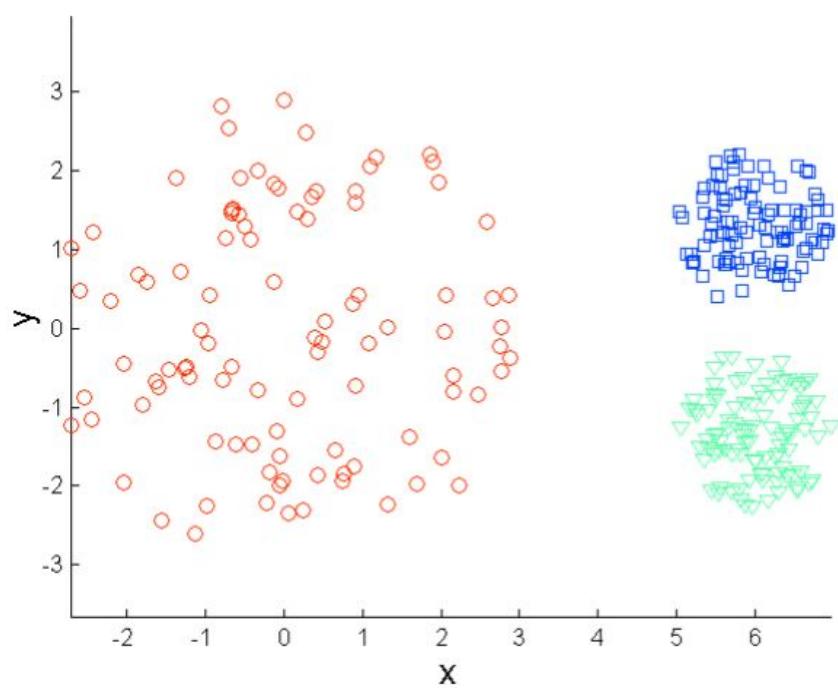
- Guaranteed to converge in a finite number of iterations
- Running time per iterations:
 - Assign data points to closest cluster center
 - $O(KN)$
 - Change the cluster center to the average of its assigned points
 - $O(N)$



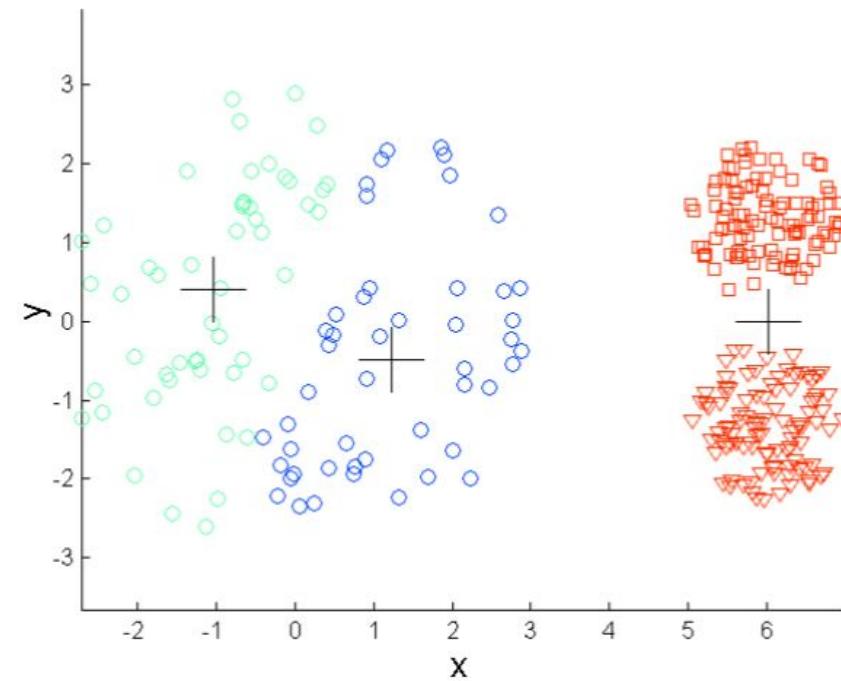
Value of K? $L(C, \mu) = \min_{\mu} \min_C \sum_{i=1}^k \sum_{x \in C_i} |x - \mu_i|^2$



Limitations: Different Sizes

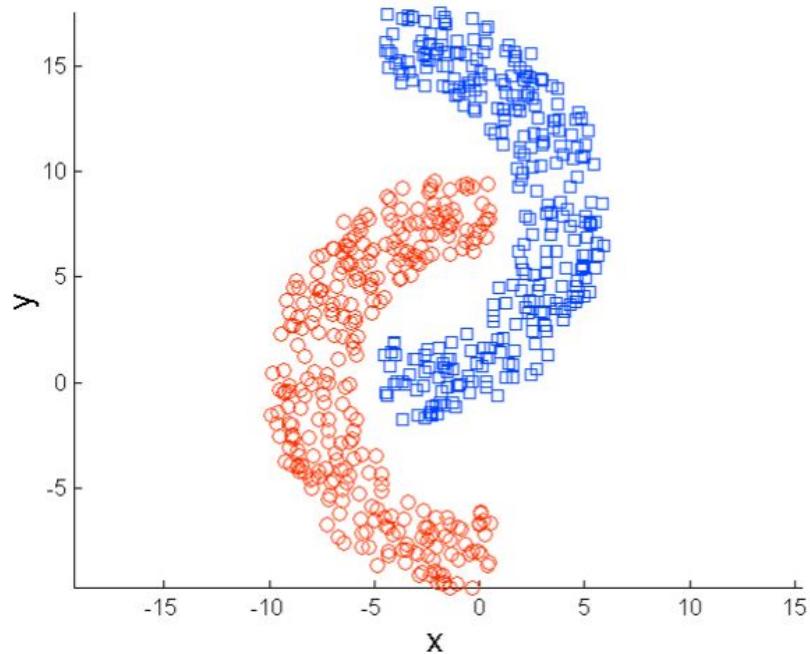


Original Points

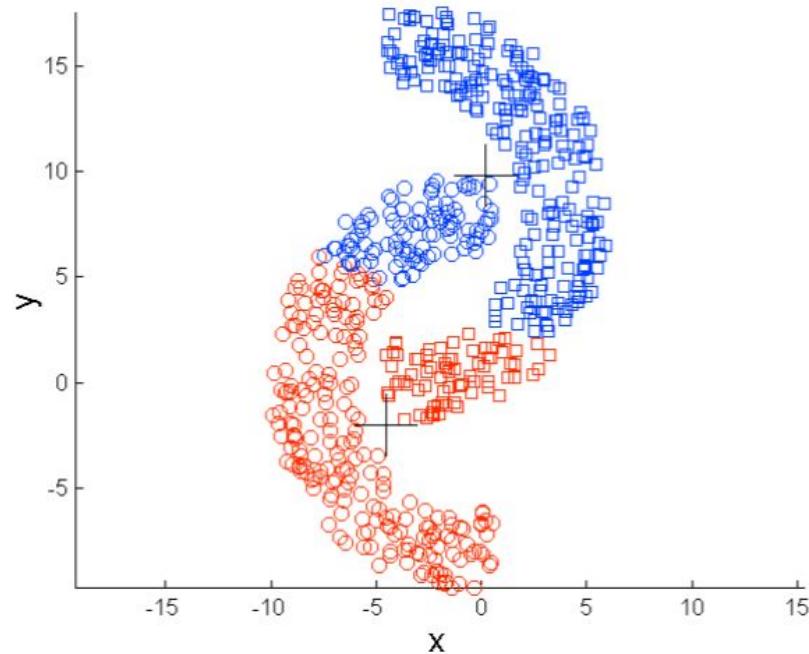


K-means (3 Clusters)

Limitations: Non-convex



Original Points



K-means (2 Clusters)

Summary: K-Means Clustering



- Strength
 - Simple, easy to implement and debug
 - Relatively efficient: $O(tkn)$, where n is # objects, k is # clusters, and t is # iterations. Normally, $k, t \ll n$.
- Weakness
 - Applicable only when mean is defined, what about categorical data?
 - Often terminates at a local optimum. Initialization is important.
 - Need to specify K , the number of clusters, in advance
 - Unable to handle noisy data and outliers
 - Not suitable to discover clusters with non-convex shapes

Breakout Room Activity



You are given a 1-d dataset as follows, $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

- a. With number of cluster, $k = 2$, and initial cluster centroids as $\{1, 2\}$, show three iterations of k-means algorithm. Use Euclidean distance function.
- b. Repeat above question with initial cluster centroids as $\{2, 9\}$. 2 point for each correct iteration.
- c. Explain your observations about how the choice of initial seed set affects the quality of results.

References



-
1. <http://www.iro.umontreal.ca/~lisa/poiteurs/kmeans-nips7.pdf>
 2. <https://cs.wmich.edu/alfuqaha/summer14/cs6530/lectures/Clustering Analysis.pdf>
 3. <https://ieeexplore.ieee.org/abstract/document/6137222>
 4. https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/Applet_KM.html
 5. <https://www.youtube.com/watch?v=4b5d3muPQmA>



Convolutional Networks



INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI



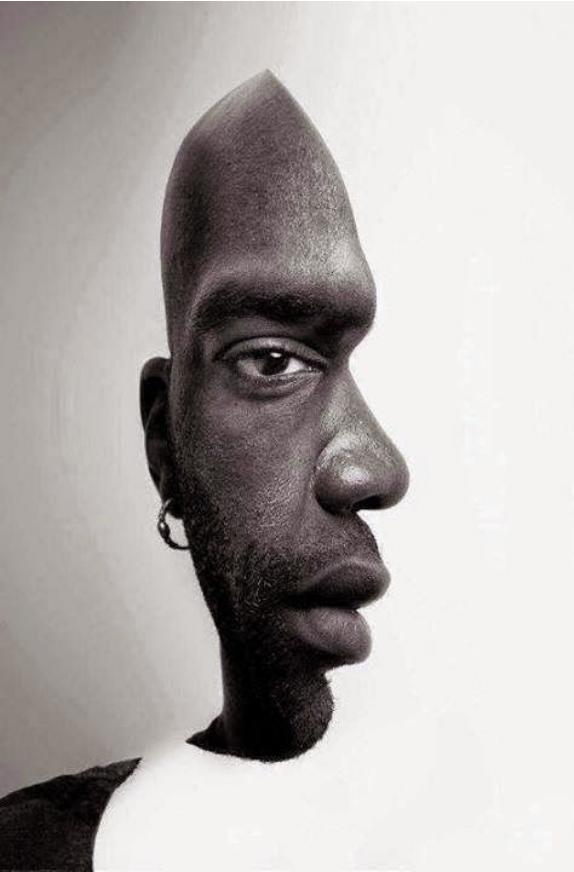
MLP: Problems



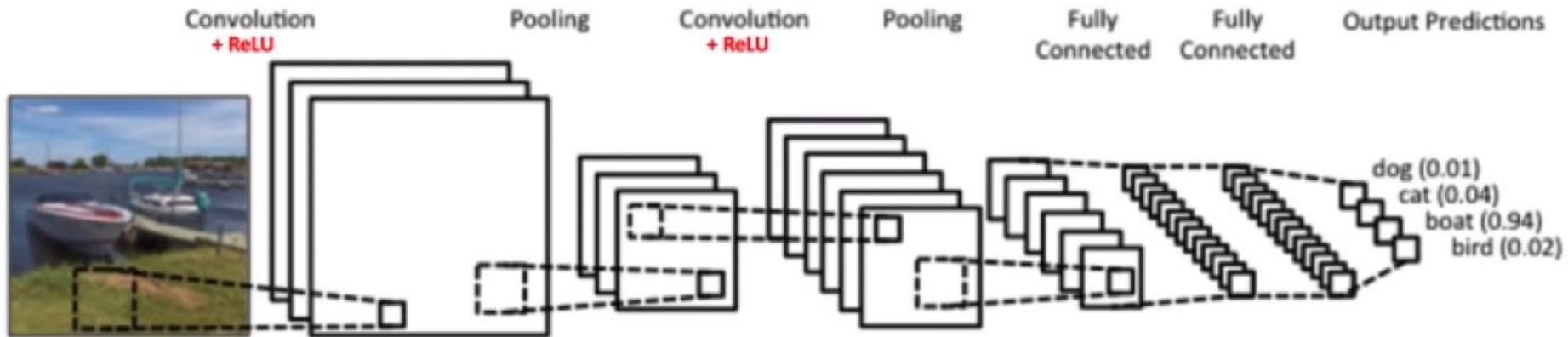
- The amount of weights rapidly becomes unmanageable for large input images
 - E.g. $224 \times 224 \times 3$ image: >150,000 weights!
- Not translation invariant
 - Reacts differently to an input's shifted version



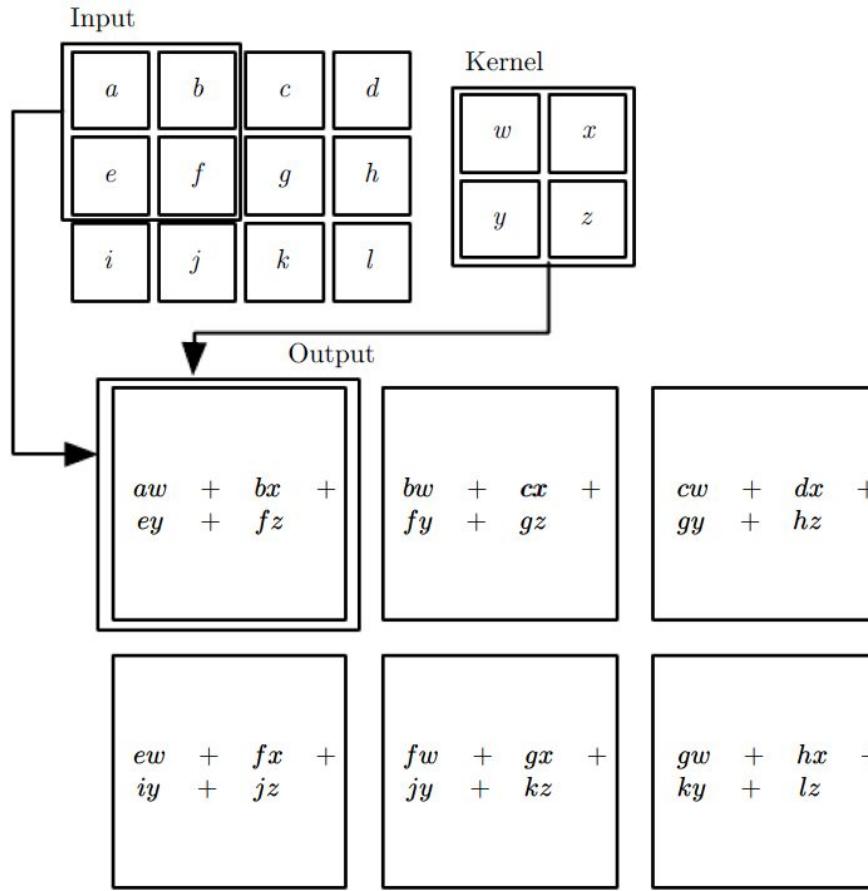
What do you see? Variance in Input Image



CNN: Architecture



Convolution Operation



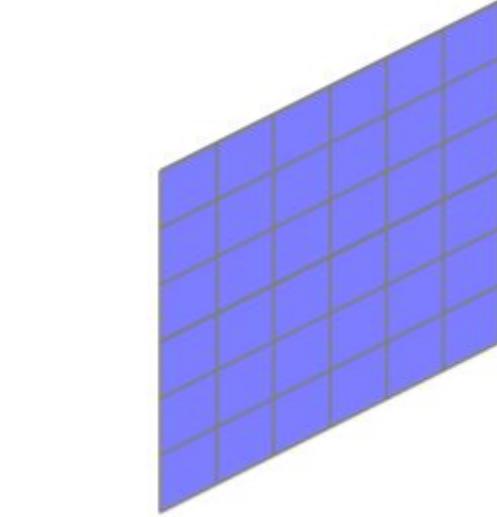
Convolution Operation



-
- $s(t) = (x * w)(t) = \sum_{a=-\infty, +\infty} x(a)w(t - a)$
 - x : Input, w : Kernel, s : Feature map
 - Two-dimensional
 - $S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n)$
 - Convolution is commutative
 - $S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$



Visual example: Convolution Operation

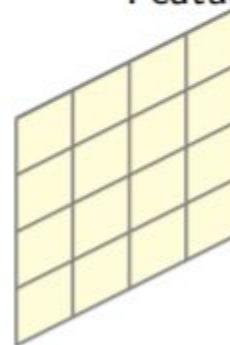


Grayscale Image

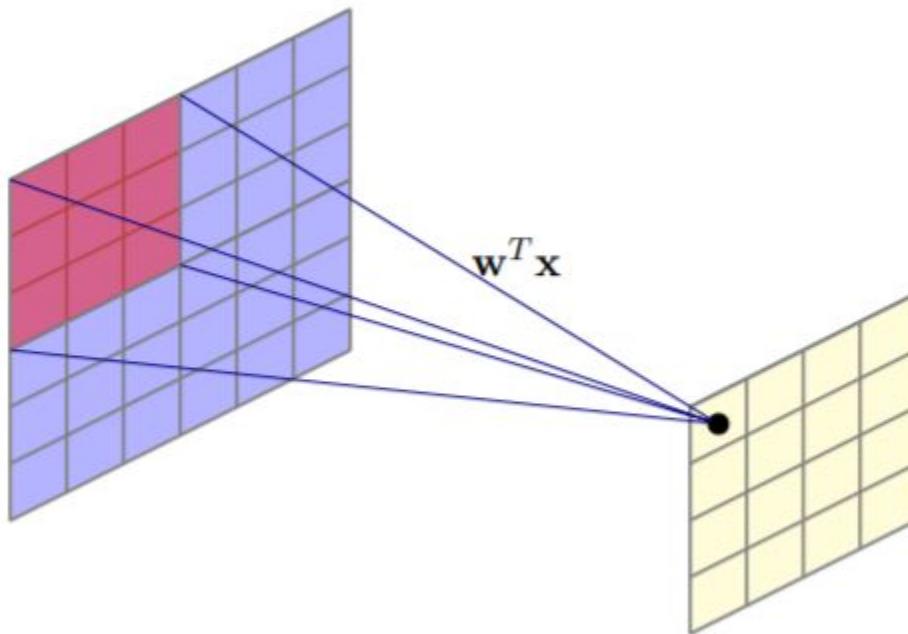
Kernel

w_7	w_8	w_9
w_4	w_5	w_6
w_1	w_2	w_3

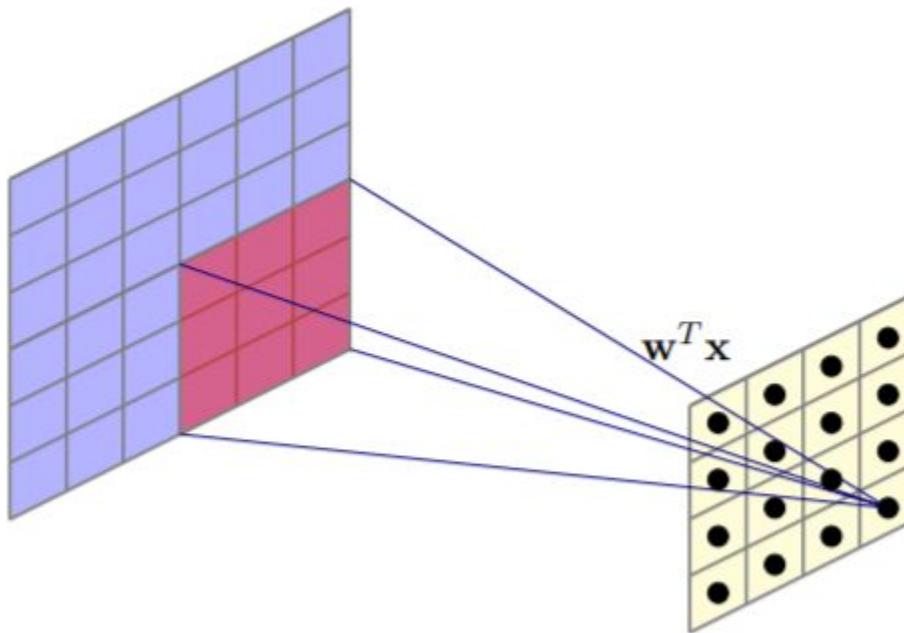
Feature Map



Visual example example: Convolution Operation



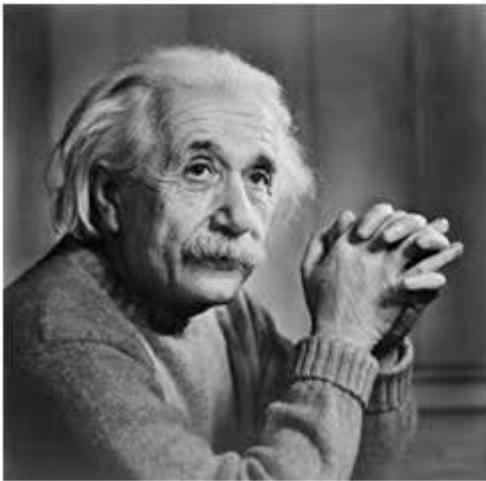
An example: Convolution Operation



Convolution: Vertical



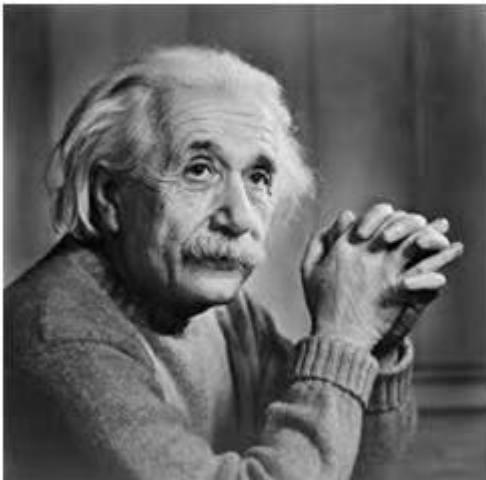
-1	0	1
-1	0	1
-1	0	1



Convolution: Horizontal



-1	-1	-1
0	0	0
1	1	1



Convolution Operation: Output Size



- For convolutional layer:
 - Suppose input is of size $W_1 \times H_1 \times D_1$
 - Filter size is K and stride S
 - We obtain another volume of dimensions $W_2 \times H_2 \times D_2$
- As before:
 - $W_2 = (W_1 - K)/S + 1$ and $H_2 = (H_1 - K)/S + 1$
- Depths will be equal



Convolution Operation: Output Size



- Output size: $(N - K)/S + 1$
 - N: input dimension
 - K: Kernel size
 - S: Stride
- In previous example:
 - N = 6, K = 3, S = 1,
 - Output size =
 - 4



Convolution: Motivation



-
- Convolution leverages four ideas that can help ML systems:
 - Sparse interactions
 - Parameter sharing
 - Equivariant representations
 - Ability to work with inputs of variable size



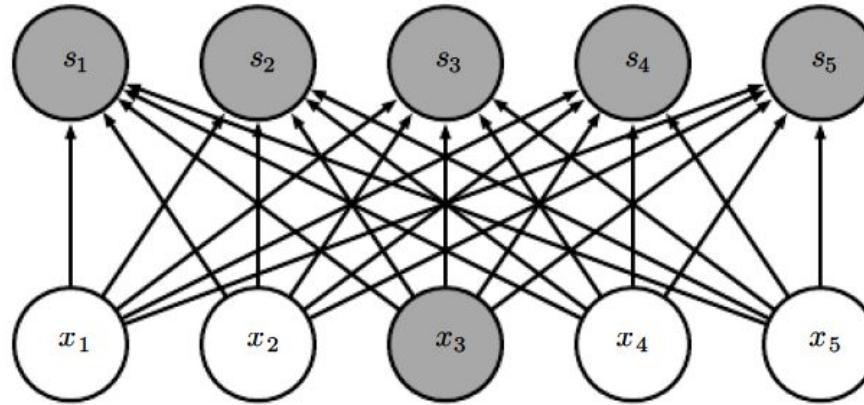
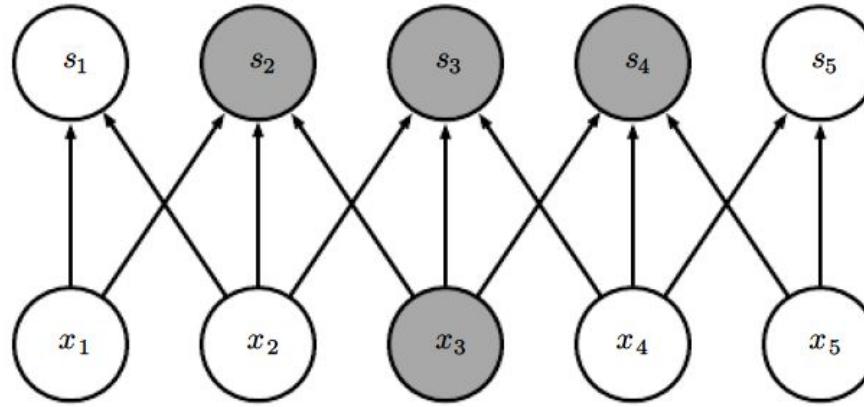
Convolution Operation: Sparse Interactions



- Sparse interactions
 - Convolutional networks typically have sparse interactions (also referred to as sparse connectivity or sparse weights)
- This is accomplished by making the kernel smaller than the input.
 - need to store fewer parameters, computing output needs fewer operations ($O(m \times n)$ versus $O(k \times n)$)



Convolution Operation: Sparse Interactions

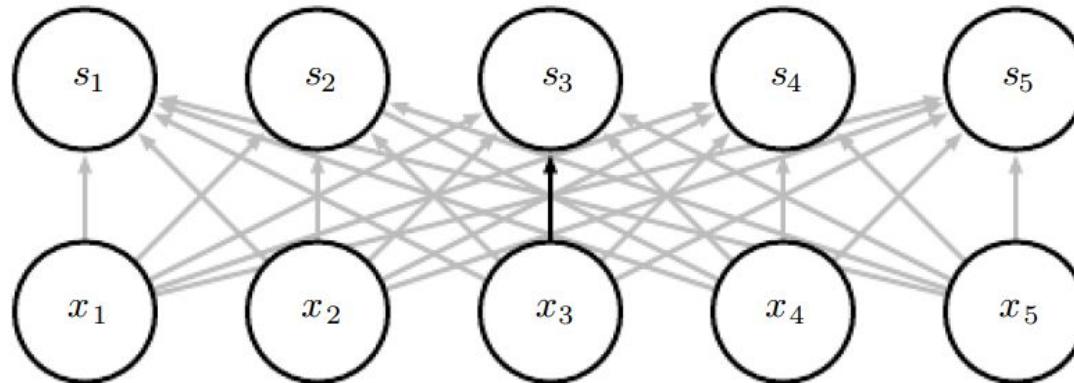
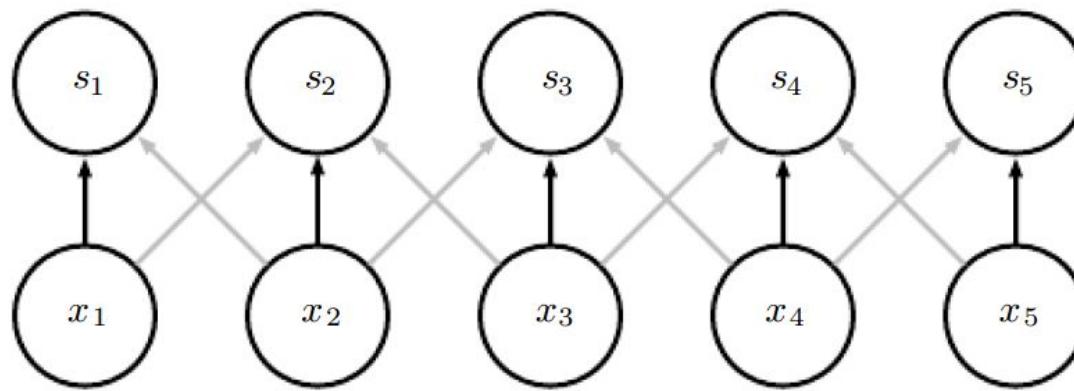


Convolution Operation: Parameter Sharing



- Parameter sharing refers to using the same parameter for more than one function in a model
 - Each member of the kernel is used at every position of the input
- Rather than learning a separate set of parameters for every location, we learn only one set.
 - This does not affect the runtime of forward propagation; it is still $O(k \times n)$, but it does further reduce the storage requirements of the model to k parameters.
 - Storage improves dramatically as $k \ll m, n$

Convolution Operation: Parameter Sharing



Convolution Operation: Equivariant representations



- To say a function is equivariant means that if the input changes, the output changes in the same way.
 - Function $f(x)$ is equivariant to a function g if
 - $f(g(x)) = g(f(x))$
- The form of parameter sharing used by CNNs causes each layer to be equivariant to translation
 - That is, if g is any function that translates the input, the convolution function is equivariant to g

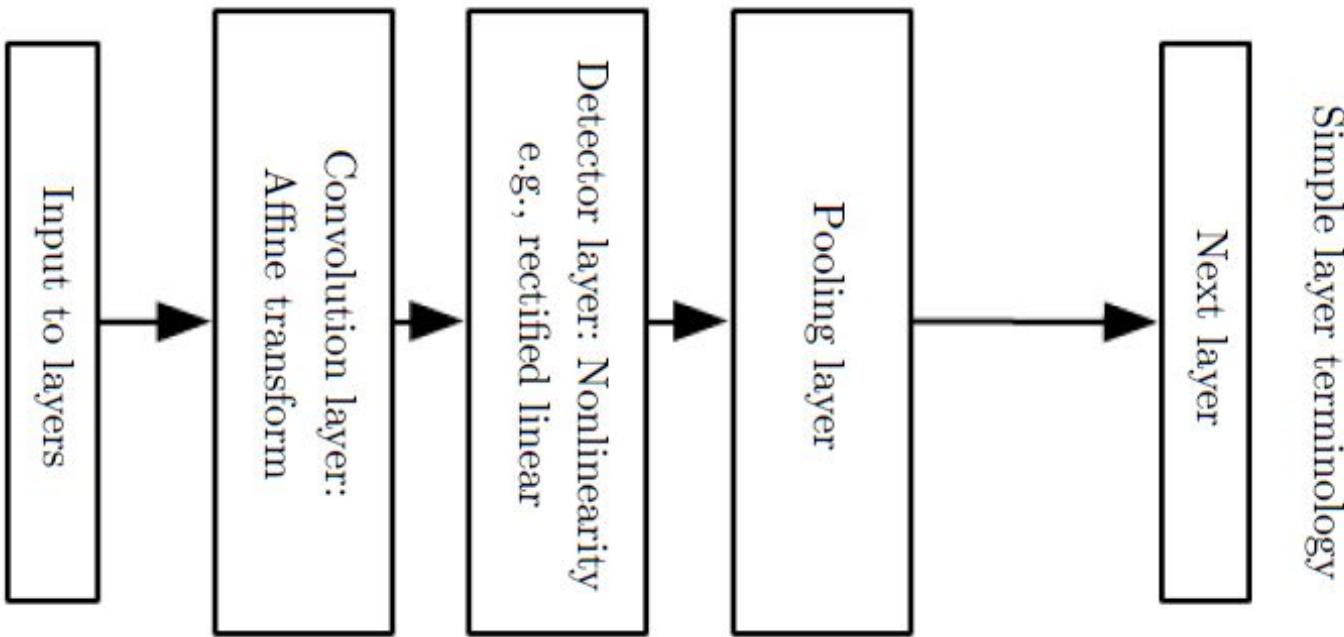
Convolution Operation: Equivariant representations



- Implication: While processing time series data, convolution produces a timeline that shows when different features appeared (if an event is shifted in time in the input, the same representation will appear in the output)
- Images: If we move an object in the image, its representation will move the same amount in the output
- This property is useful when we know some local function is useful everywhere (e.g. edge detectors)
- Convolution is not equivariant to other operations such as change in scale or rotation

Pooling

- Linear Activations [Convolution] -> Non-linear Activations [Detector] -> Pooling

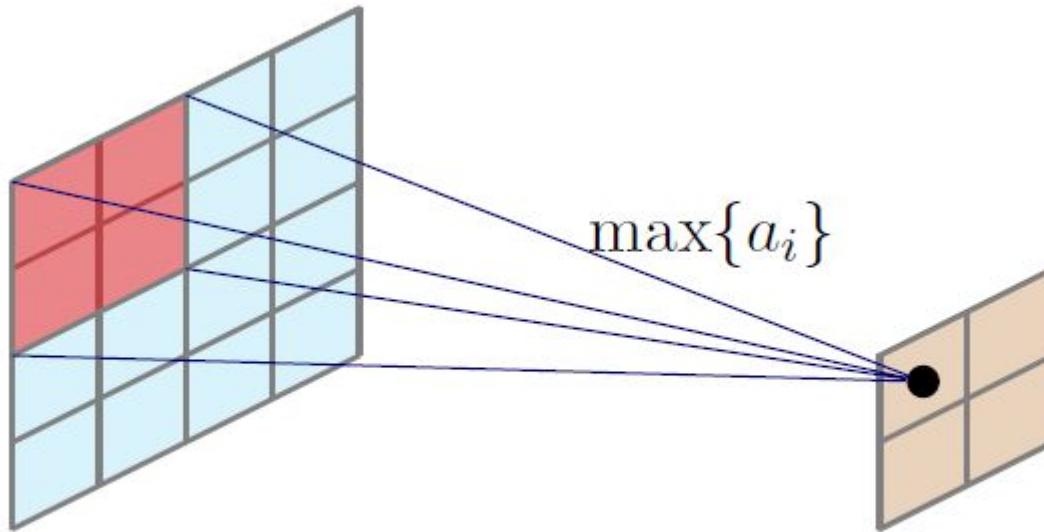


Pooling

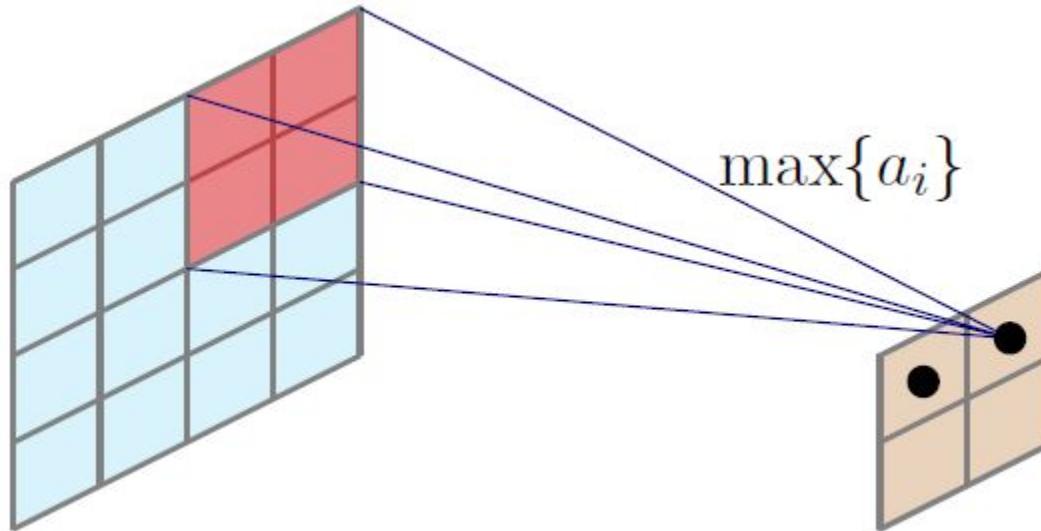


- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
 - Max pooling operation reports the maximum output within a rectangular neighborhood.
- Other popular pooling functions include
 - the average of a rectangular neighborhood,
 - the L₂ norm of a rectangular neighborhood, or
 - a weighted average based on the distance from the central pixel
- In all cases, pooling helps to make the representation become approximately invariant to small translations of the input.

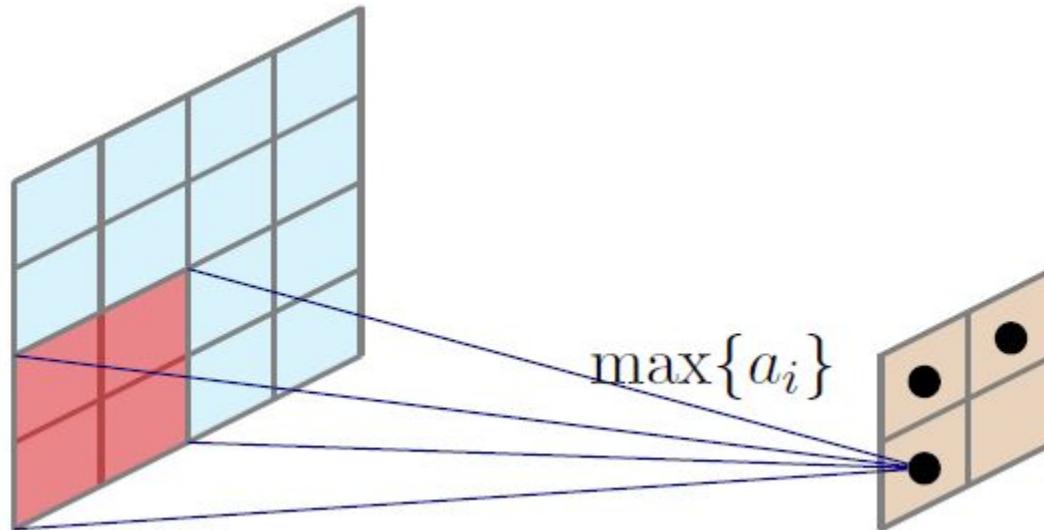
Pooling: Max Pooling



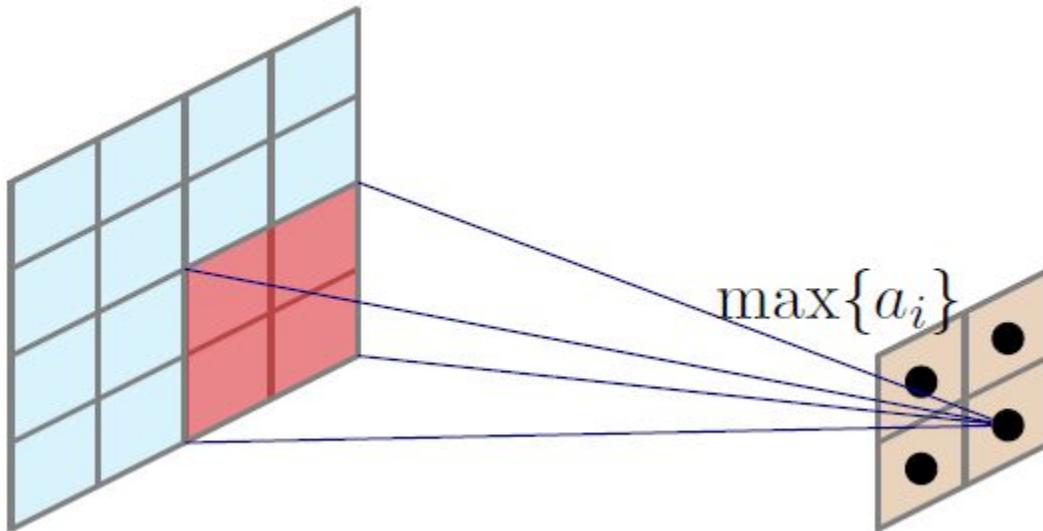
Pooling: Max Pooling



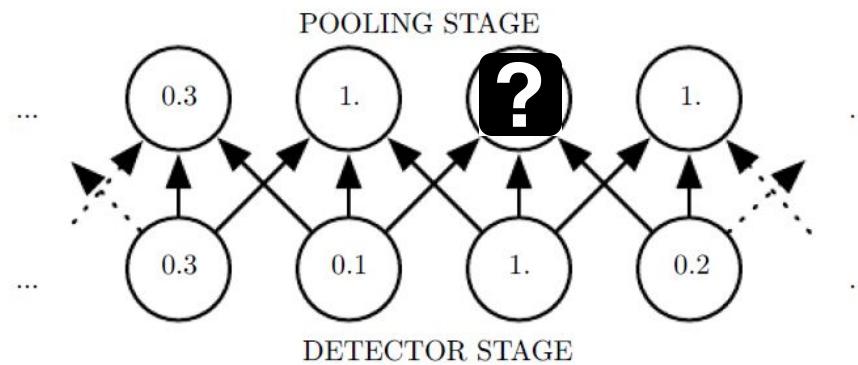
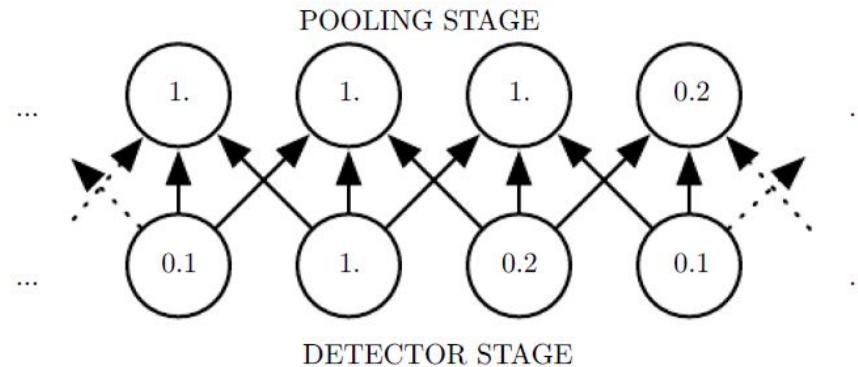
Pooling: Max Pooling



Pooling: Max Pooling



Pooling

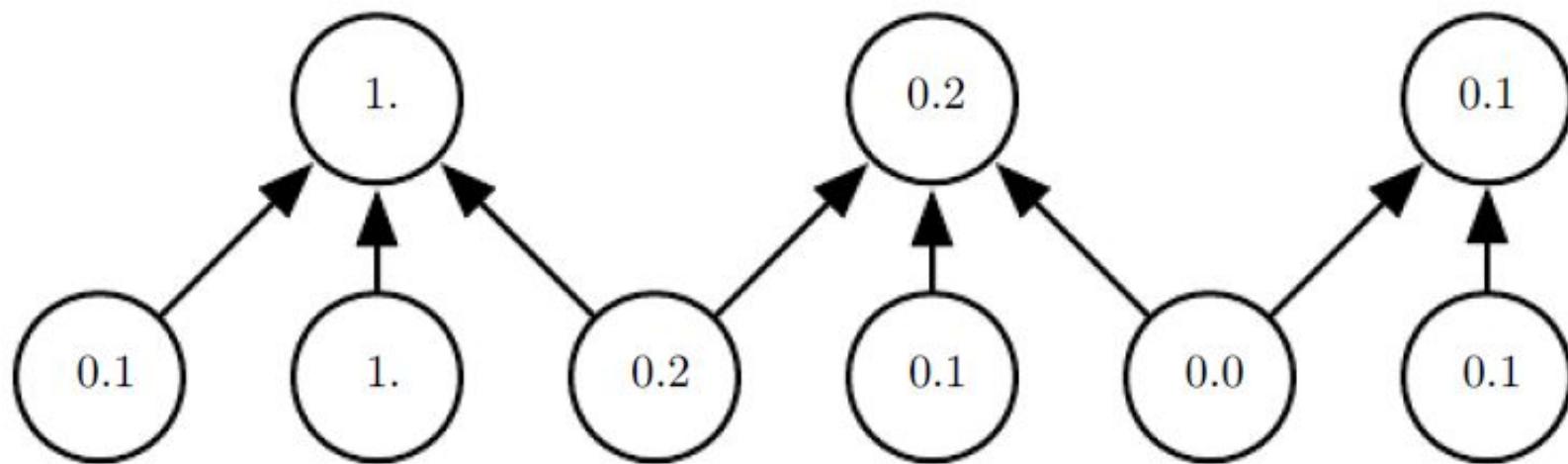


Pooling

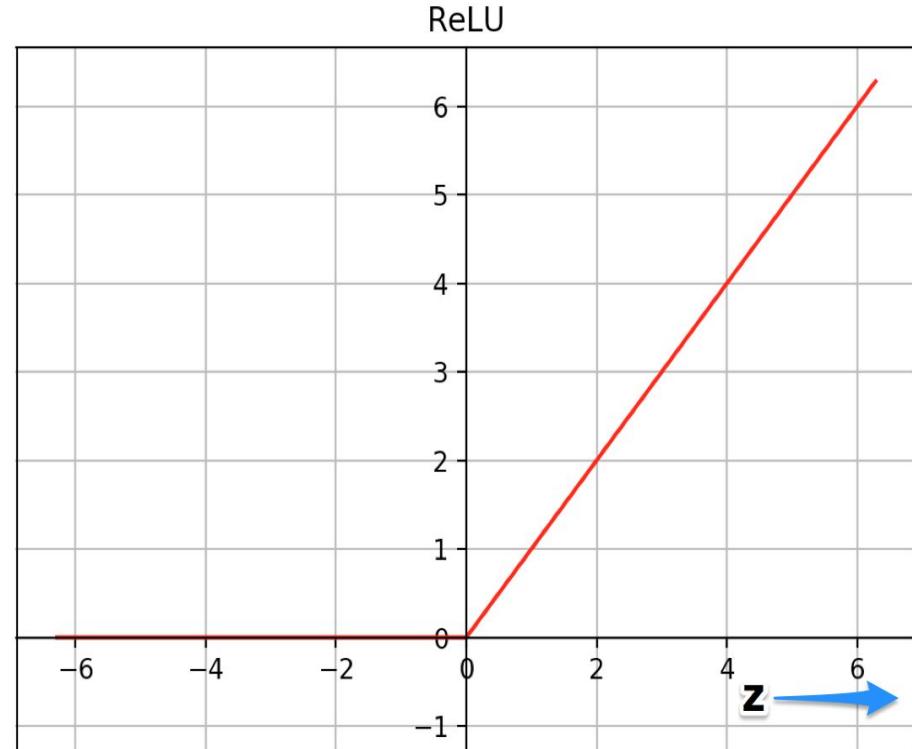
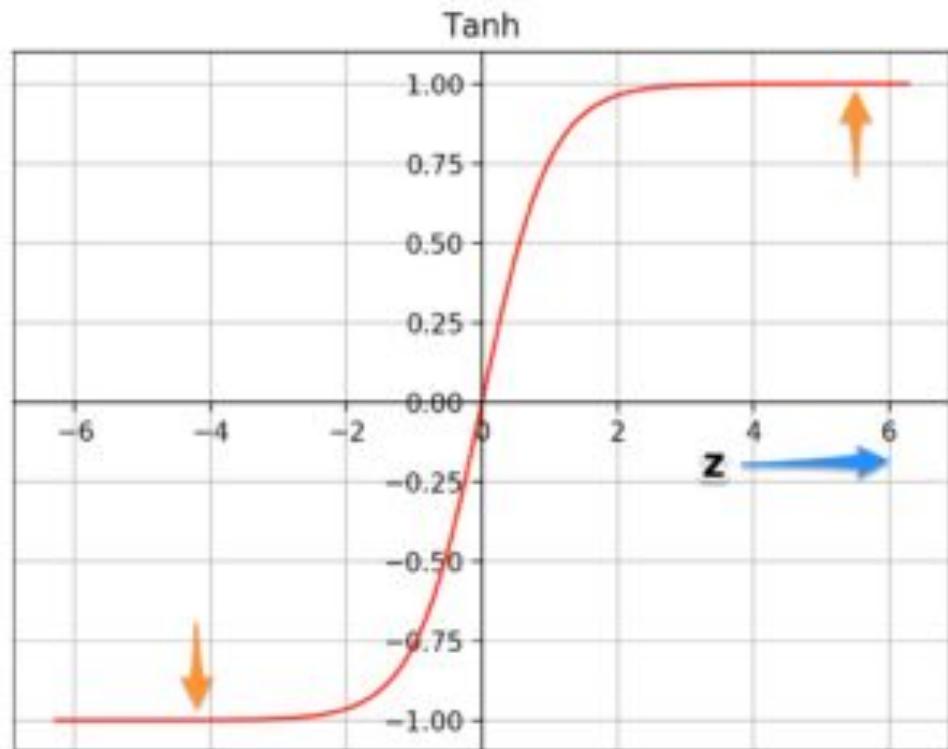


- Invariance to local translation can be a very useful property if we care more about *whether some feature is present* than *exactly where it is*
 - For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy.
- In other contexts, it is more important to preserve the location of a feature.
 - For example, if we want to find a corner defined by two edges meeting at a specific orientation
- Since pooling is used for downsampling, it can be used to handle *inputs of varying sizes*

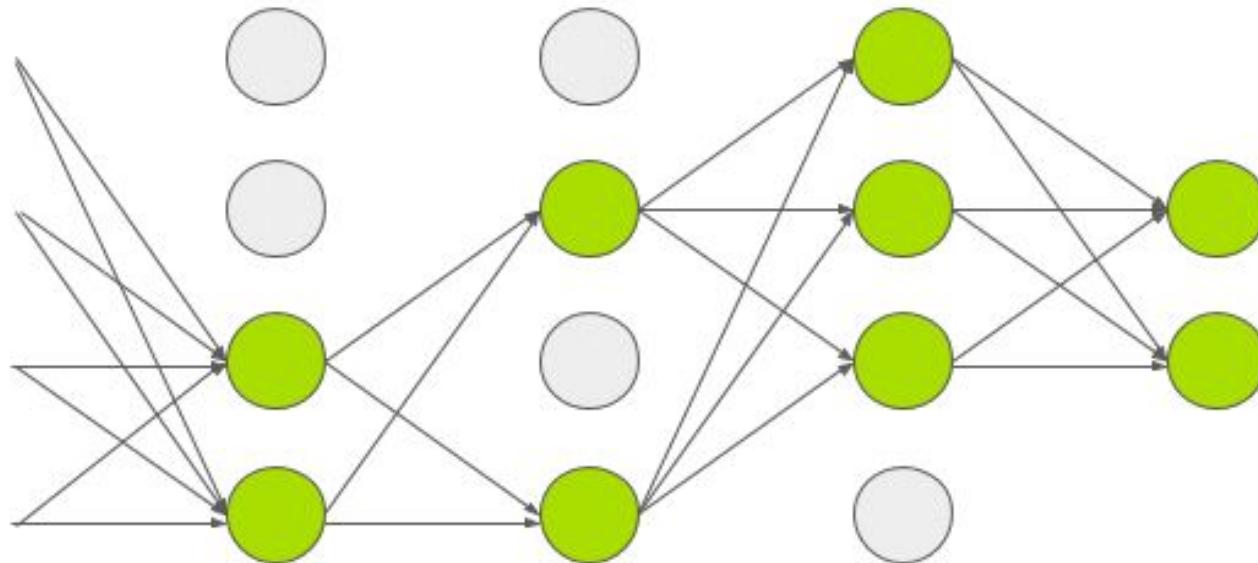
Pooling: Downsampling



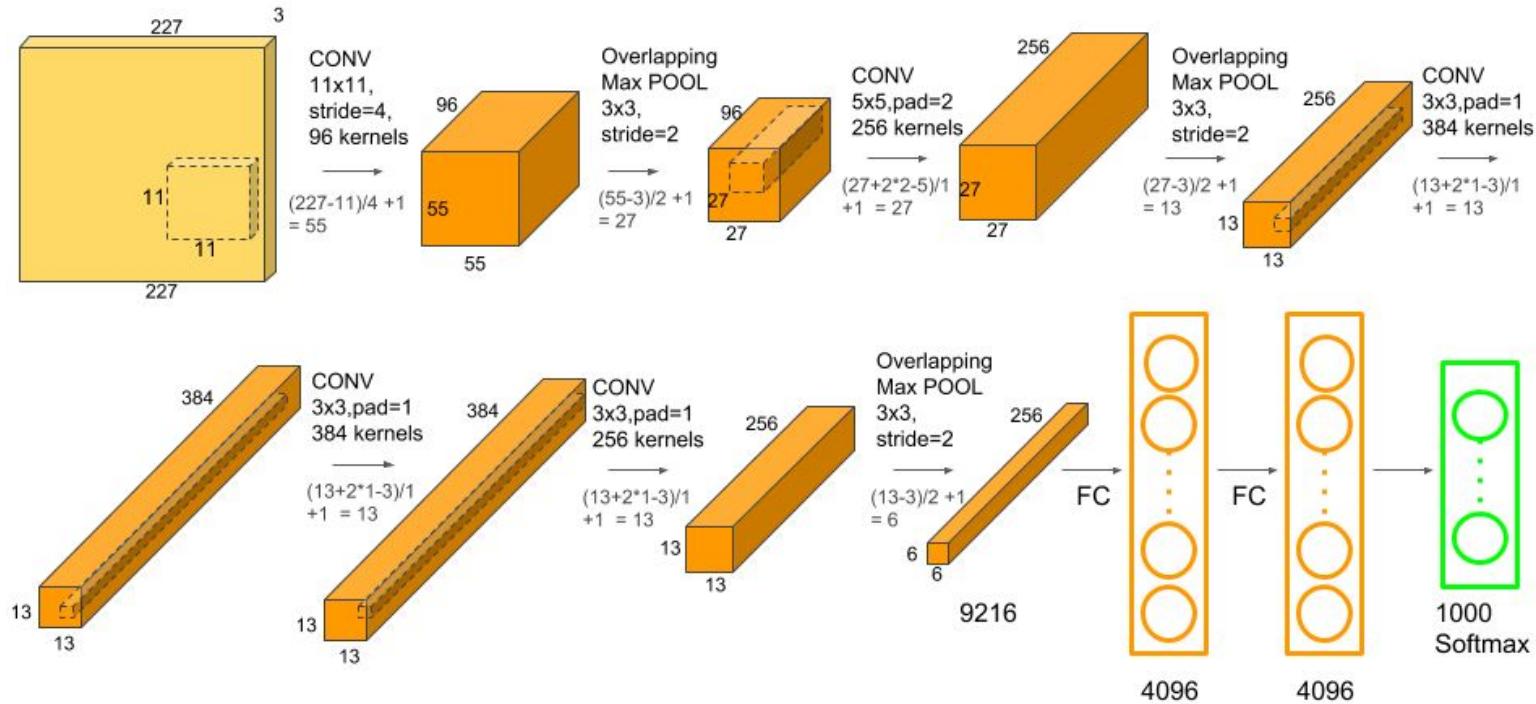
ReLU: Rectified Linear Unit



Dropout



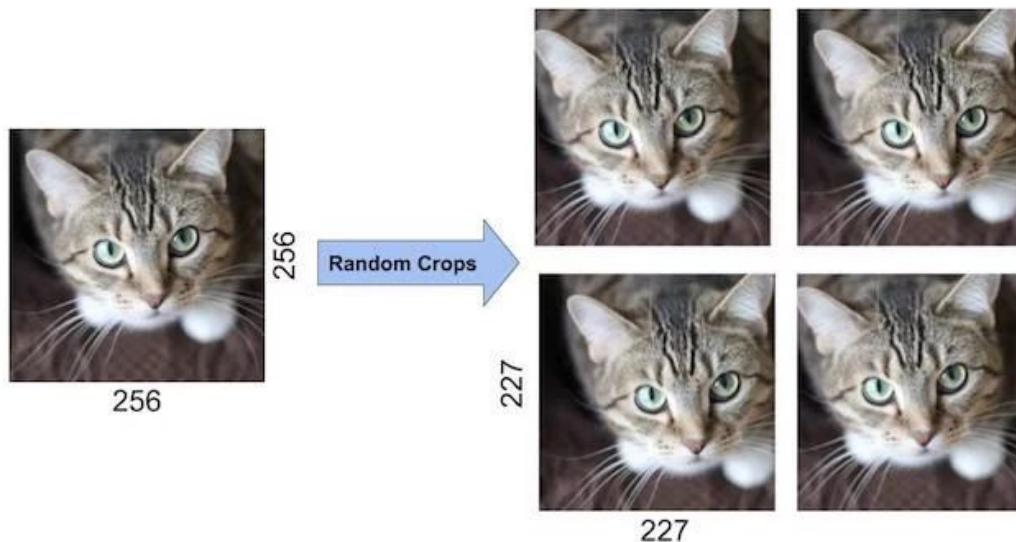
Walk Through: AlexNet



AlexNet



- The input to AlexNet is an RGB image of size 256×256
- Random crops of size 227×227 were generated from inside the 256×256 images to feed the first layer of AlexNet.
- Input: $227 \times 227 \times 3$



AlexNet



- 5 Convolutional Layers and 3 Fully Connected Layers
- It has 60 million parameters and 650,000 neurons and took five to six days to train on two GTX 580 3GB GPUs!
- Input: $227 \times 227 \times 3$
- The first Conv Layer of AlexNet contains 96 kernels of size $11 \times 11 \times 3$.
 - Width and height of output: $(227 - 11)/4 + 1 = 55$
- Number of parameters in first layer?
 - $(11 \times 11 \times 3 + 1) * 96 = 34,9444$
- Number of Computations: $34,9444 \times 55 \times 55 = 10,57,05,600$
- <https://airtable.com/shrArXKRCAu4KhAwZ/tbloN5WYjFYpKGUEt?blocks=hide>

References



1. Chapter 9, Deep Learning: Ian Goodfellow, Yoshua Bengio, Aaron Courville (<http://www.deeplearningbook.org/>)
2. <https://neurdiness.wordpress.com/2018/05/17/deep-convolutional-neural-networks-as-models-of-the-visual-system-qa/>
3. <https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf>
4. <https://www.youtube.com/watch?v=2-Ol7ZBoMmU>
5. <https://www.youtube.com/watch?v=ZOXOwYUVCqw>

