Course Logistics

AIM OF THE COURSE

1. Design, implementation & applications of supervised and unsupervised ML algorithms
2. Classification algorithms: Perceptron, Logistic Regression, SVM, Neural Networks, Decision Trees, Ensembles

COURSE STRUCTURE AND EVALUATION

36-42 lectures or 12-14 weeks (3 hrs per week Mon/Wed/Fri 9am to 10am) at F106

Component	Duration	Weightage (%)	Date & Time	Nature of Component
Mid Term Exam	1.5 hours	30	9/10 FN	Closed Book
Assignments	--	25	TBA	Open Book
Class Participation		10	TBA	Open Book
Comprehensive Exam	3 Hours	35	6/12 AN	Closed Book

REFERENCES

1. Pattern Recognition and Machine Learning, Christopher M Bishop
2. Machine Learning, Tom M Mitchell
3. The Elements of Statistical Learning, Trevor Hastie et. al
4. Python Machine Learning, Sebastian Raschka

OBJECTIVES

1. Learn theoretical and practical aspects of linear models for regression & classification
2. Understand probabilistic, discriminative & generative models for classification
3. Learn theoretical and practical aspects of SVM and ANN
4. Understand decision tree learning and ensemble methods

COURSE PLAN:

Lecture No.	Learning objectives	Topics to be covered	Chapter in the Text Book
1 – 2	To introduce the course	Course Introduction & Motivation	Lecture Slides
3 - 6	To understand linear models for classification	Linear Regression, Polynomial regression	T1 – Ch. 1.1 T1 – Ch. 3.1
7 – 13	To understand linear models for classification	Discriminant functions, Least squares for classification, perceptron algorithm	T1 – Ch. 4.1
14 - 20	To understand probabilistic generative and discriminative models	Probabilistic generative models – Maximum likelihood solution, Naïve Bayes classifier, probabilistic discriminative models – Logistic Regression	T1 – Ch.4.2 T1 – Ch. 4.3 T2 – Ch 6
21 – 26	To understand ANN	Feed forward Neural Networks, Backpropagation	T1 – Ch. 5.1, 5.2, 5.3 T2 – Ch. 4
27 – 32	To understand SVM	Maximum margin classifiers	T1 – Ch. 7.1
33 - 34	To understand unsupervised learning	Clustering and Expectation Maximization	T1 – Ch. 9.1, 9.2
35 – 36	To understand decision tree learning	Decision Tree Learning	T2 – Ch. 3
37 – 40	To understand ensemble methods	Bias Variance tradeoff, Bagging and Boosting	T1 – Ch.14.2, 14.3

Essential Readings

- CS229 Lecture Notes, Andrew Ng and Tengyu Ma
  - Linear regression - Chapters 1.1, 1.2, 1.3
  - Support Vector Machines - Chapter 6
- Pattern Recognition and Machine Learning, Christopher M Bishop
  - Introduction - Chapters 1.1, 1.2
  - Linear Models for Regression - Chapters 3.1
  - Linear Models for Classification - Chapters 4.1, 4.2, 4.3
  - Neural Networks - Chapters 5.1, 5.2, 5.3
- Introduction to Data Mining, Pang Ning Tan, Michael Steinbach, Vipin Kumar
  - Classification: Basic Concepts - Chapters 4.3, 4.4
  - Classification: Alternative Techniques - Chapters 5.4, 5.5
- STAT 479 Lecture Notes, Sebastian Raschka
  - Ensemble Methods - Chapter 7

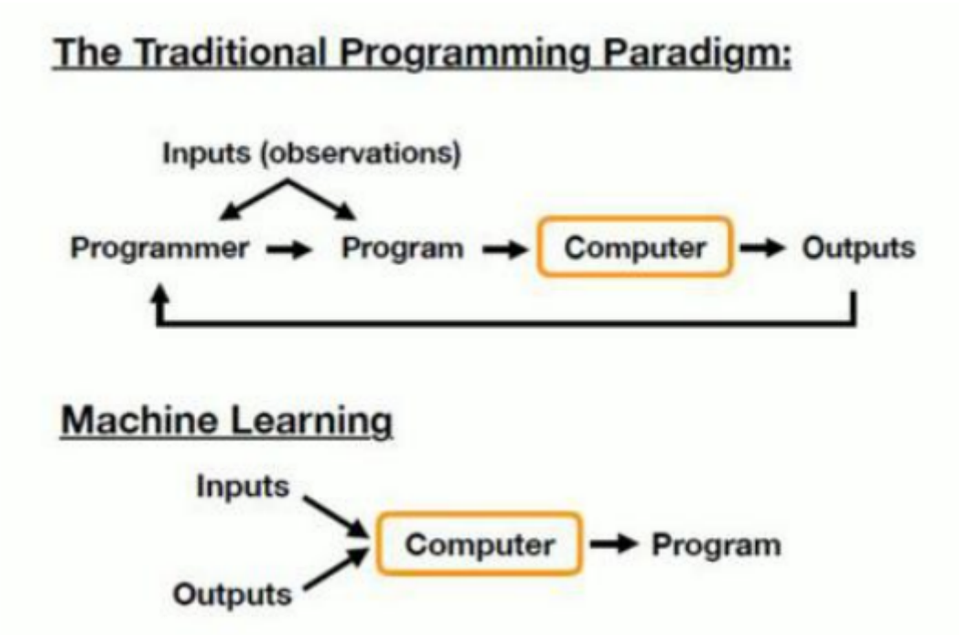
Chapter-1 Models for Regression

1.1 Definitions, Applications and Types of ML

For many problems, it's difficult to program the correct behavior by hand
- recognizing people & objects
- understanding human speech

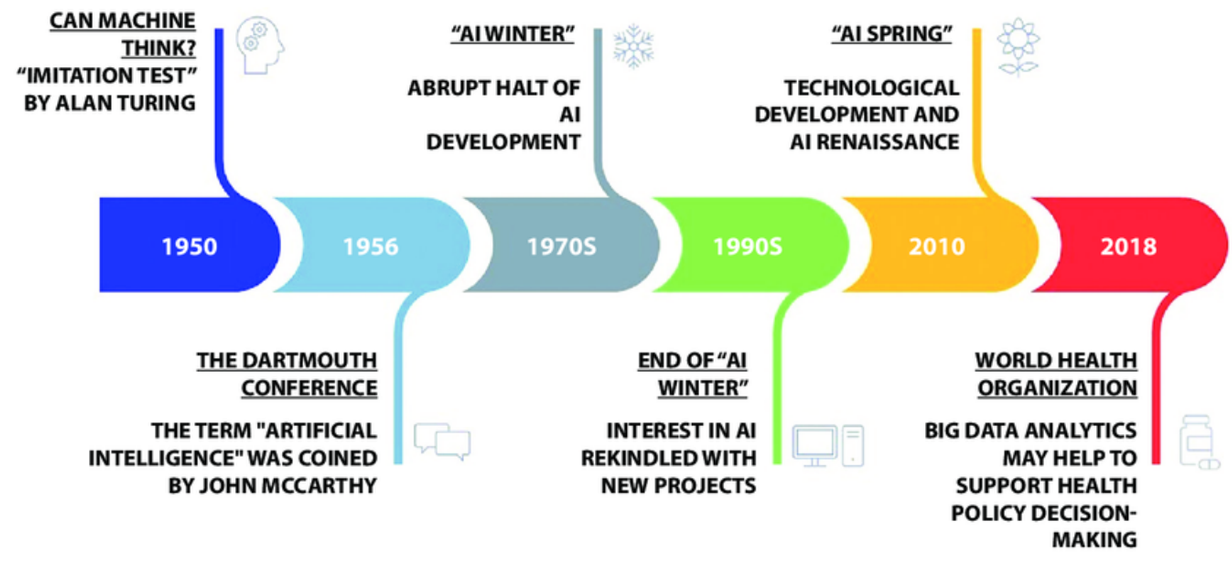
ML approach - program an algorithm to automatically learn from data or experience

We ask the question "How can we build computer systems that automatically improve with experience and what are the fundamental laws that govern all learning processes?"



- The term Machine Learning was first coined by Arthur Lee Samuel in 1959.
- He defined ML as follows:
  - ML is the field of study that gives computers the ability to learn without being explicitly programmed.

TIMELINE DIAGRAM OF ARTIFICIAL INTELLIGENCE HISTORY



A FORMAL DEFINITION

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Example: Handwritten Digit Recognition

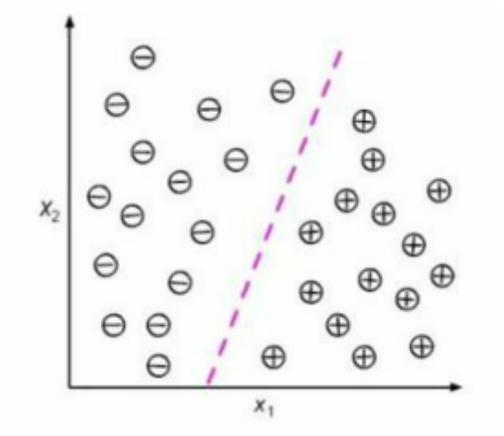
- T: Classifying handwritten digits from images
- P: Percentage of digits classified correctly
- E: Dataset of digits given classification (MNIST)

APPLICATIONS

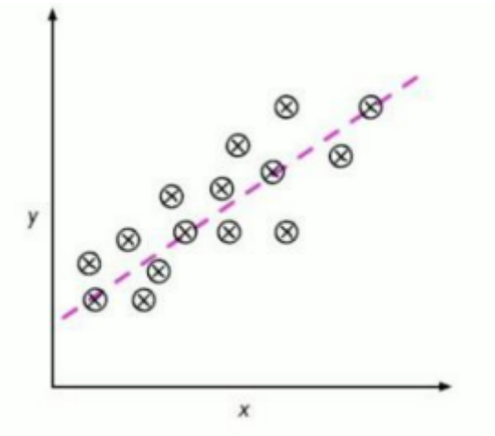
- Personal Assistants
- Advertising & Business Intelligence
- News Feeds/ Filtering Algorithms
- Recommendation Engines
- ....and so on

CATEGORIES OF ML

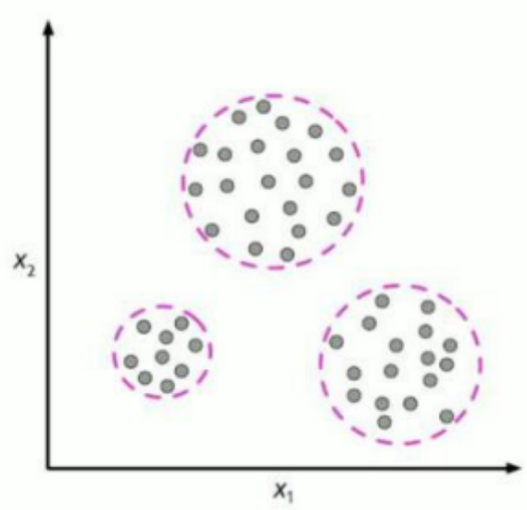
1. Supervised Learning - Labeled Data, Direct Feedback, Predict Outcome/Future
2. Unsupervised Learning - Unlabeled Data, No Feedback, Find Hidden Structure in Data
3. Reinforcement Learning - Decision Process, Reward System, Learn Series of Actions



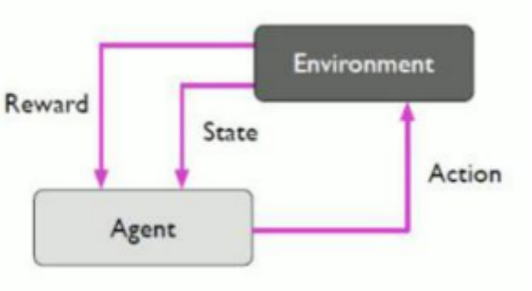
Classification: Predict likelihood that something belongs to some class. E.g. Spam E-Mail Detection



Regression: Predicts a numerical value. E.g. Predict amount of rain on a certain day.



Clustering: Forms clusters based on similarity and finds patterns in data. E.g. Fake News Detection



Reinforcement Learning: Make predictions by getting rewards or penalties based on actions performed in an environment. E.g. Robotics

1.2 Supervised Learning

Goal: Given a training set, learn a function

$$h : X \to Y$$

so that h(x) is a good predictor for y. This function is called a hypothesis.

For a classification task, we define the hypothesis as

$$h : X \to Y$$

where

$$X = \mathbb{R}^m, Y = \{1, \ldots, K\} \; with \; class \; labels \; K$$

For binary classification, we consider

$$Y = \{0, 1\} \; or \; \{-1, 1\}$$

For regression, we define our hypothesis as

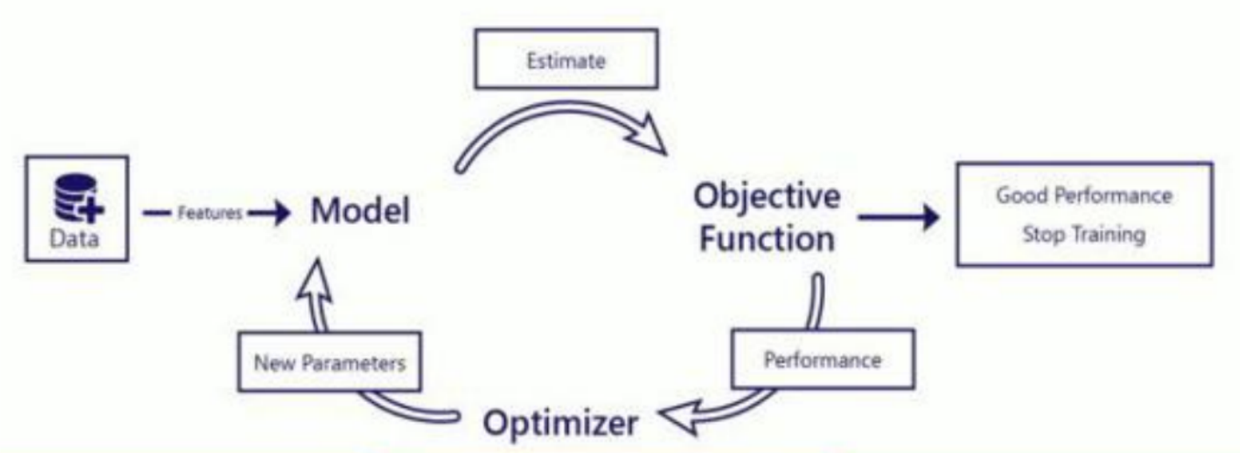
$$h : \mathbb{R}^m \to \mathbb{R}$$

Algorithm Categorization Schemes

The supervised learning algorithms can be categorized as follows:

1. Eager vs Lazy - process data immediately vs defer the processing until the prediction
2. Batch vs Online - learn on the entire data at once vs learn from one example at a time.
3. Discriminative vs Generative - probabilistic vs direct

Modular Approach

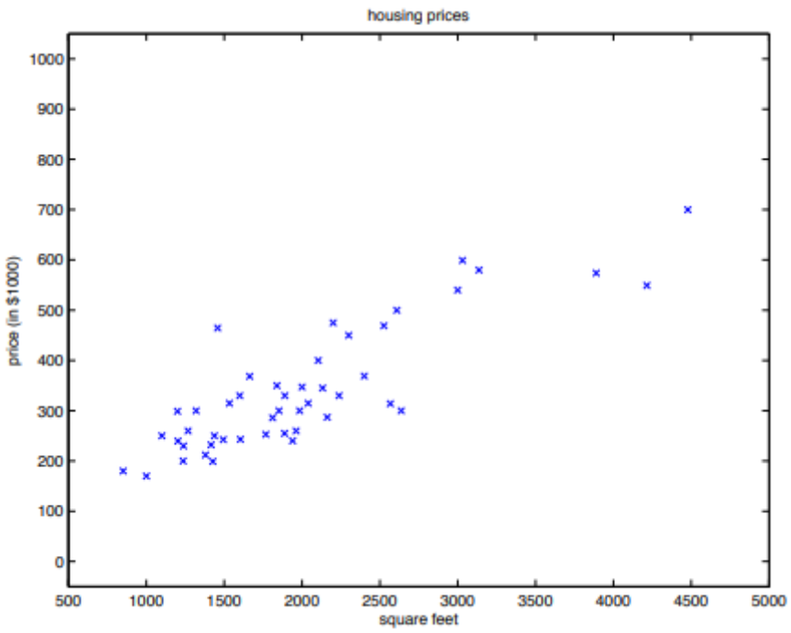


- Data is the driving force
- Model is the mathematical relationship from input features to output labels
- Loss function is the difference between predicted and actual values
- Optimization so that the model gets updated to get the best solution
- Evaluation to determine the performance of the model

1.3 Regression Analysis - House Price Prediction

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

Plot of the data



Question: Given data like this, how can we learn to predict the prices of other houses, as a function of the size of their living areas?

Notation

$$x^{(i)} \rightarrow input\ features$$

$$y^{(i)} \rightarrow output\ features$$

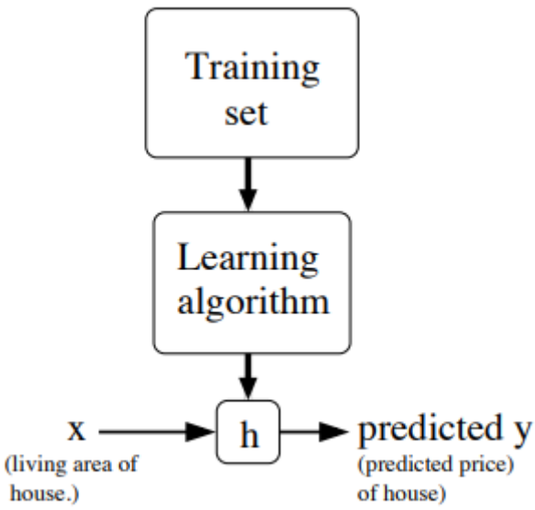
$$(x^{(i)}, y^{(i)}) \rightarrow training\ example$$

$$\{(x^{(i)}, y^{(i)}) : i = 1, \ldots, n\} \rightarrow training\ set$$

Given a training set we wish to learn the hypothesis

$$h : X \rightarrow Y$$

so that h(x) is a good predictor for the corresponding value of y.



Example: We consider two input features, living area and number of bedrooms and wish to predict price of the house.

Living area ($feet^2$)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

- The x are two-dimensional vectors in R^2
- $x_1^{(i)}$ denotes living area of i^{th} house
- $x_2^{(i)}$ denotes number of bedrooms of i^{th} house

We try to approximate y as a linear function of x

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Here the θ's are parameters or weights that parameterize the space of linear functions mapping from X to Y.

To simplify the notation we set $x_0 = 1$ and rewrite the hypothesis as

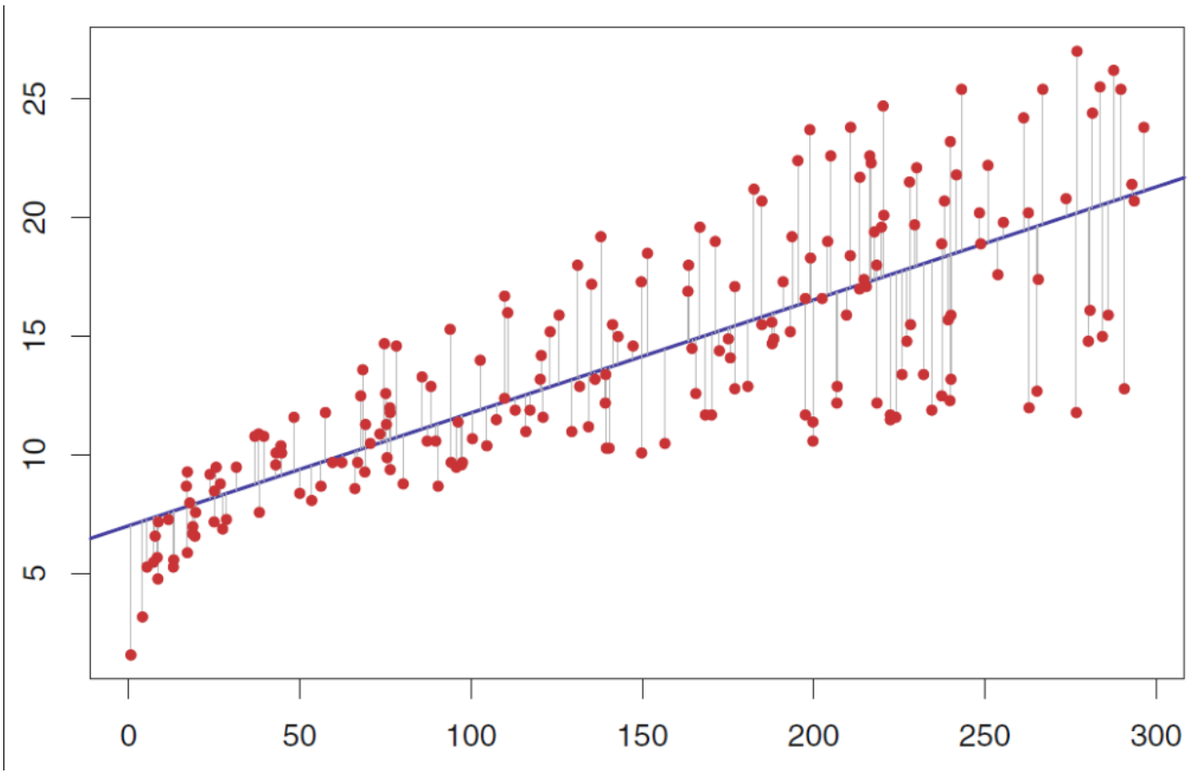
$$h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T x$$

A Simpler Model

Essentials of Linear Algebra - Cheat Sheet by Ivan Savov

- A single feature x_1

$$y = \theta_0 + \theta_1 x_1$$



- How do we learn model parameters?
  - make h(x) close to y, at least for the available training examples.
  - define a function that measures, for each value of the θ's, how close the $h(x^{(i)})$'s are to the corresponding $y^{(i)}$'s.

Cost function:

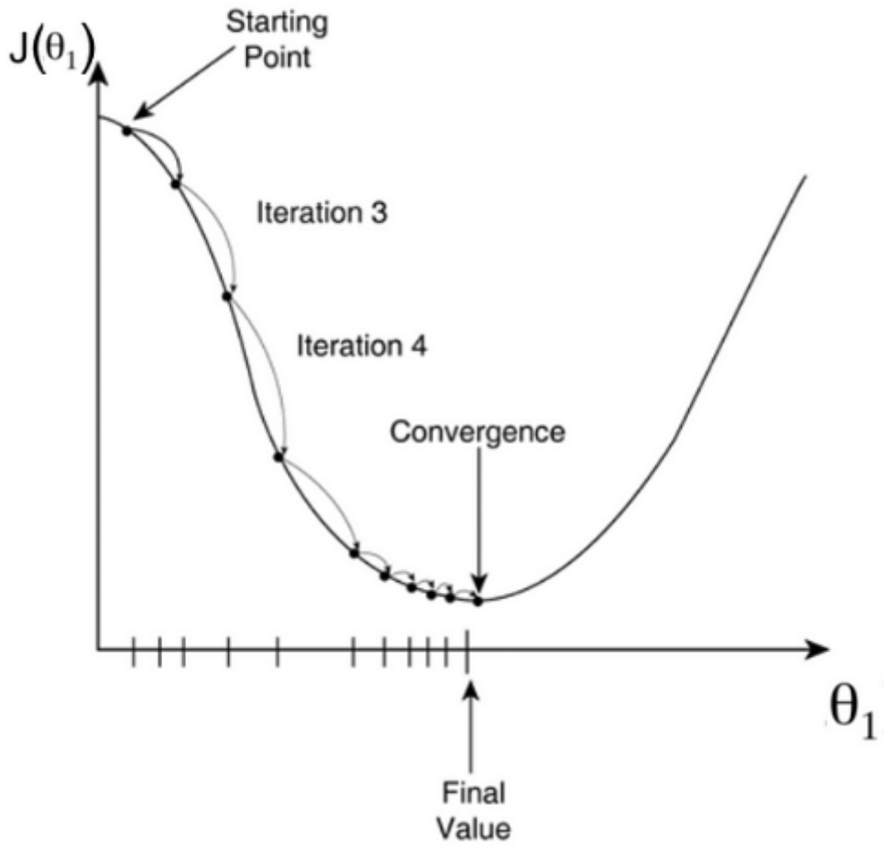
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

A cost function is used to gauge the performance of the ML model; compares the predicted values with actual values.

1.4 Least Mean Squares (LMS) Algorithm

- Want to choose θ so as to minimize J(θ)
- Start with an initial guess for θ and repeatedly change θ to make J(θ) smaller, until convergence to a value of θ that minimizes J(θ)

Gradient Descent Algorithm



- A generic optimization algorithm that can be used to learn weight vectors of most of the ML models.
- The gradient always points in the direction of steepest increase in the loss function. The algorithm takes a step in the direction of the negative gradient to reduce the loss.

We define the LMS update rule as follows:

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

Now we compute the partial derivative in the RHS:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y\right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \left(h_\theta(x) - y\right) x_j
\end{aligned}$$

Hence, for a single training example, the update rule takes the form:

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

This rule is called the LMS or Widrow-Hoff learning rule.

For more than one training example, we replace the rule with the following algorithm:

$$
\begin{aligned}
&\text{Repeat until convergence } \{ \\
&\quad \theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)} \qquad \text{(for every } j\text{).} \\
&\}
\end{aligned}
$$

This is known as **Batch Gradient Descent**. It looks at every example in the entire training set on every step.

Another alternative is to use the following algorithm:

$$
\begin{aligned}
&\text{Loop } \{ \\
&\quad \text{for i=1 to m, } \{ \\
&\quad\quad \theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)} \qquad \text{(for every } j\text{).} \\
&\quad \} \\
&\}
\end{aligned}
$$

This is known as **Stochastic Gradient Descent**. We repeatedly run through the training set, and each time we encounter a training example, we update the parameters according to the gradient of the error with respect to that single training example only.

1.5 Normal Equations

- A second way to minimize J.
- Explicitly take the derivatives with respect to θ_j's and set them to zero.

Since we know

$$h_\theta(x^{(i)}) = (x^{(i)})^T \theta$$

it can be shown that

$$
\begin{aligned}
X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \\
&= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(m)}) - y^{(m)} \end{bmatrix}.
\end{aligned}
$$

$$
\begin{aligned}
\frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) &= \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \\
&= J(\theta)
\end{aligned}
$$

To minimize J,

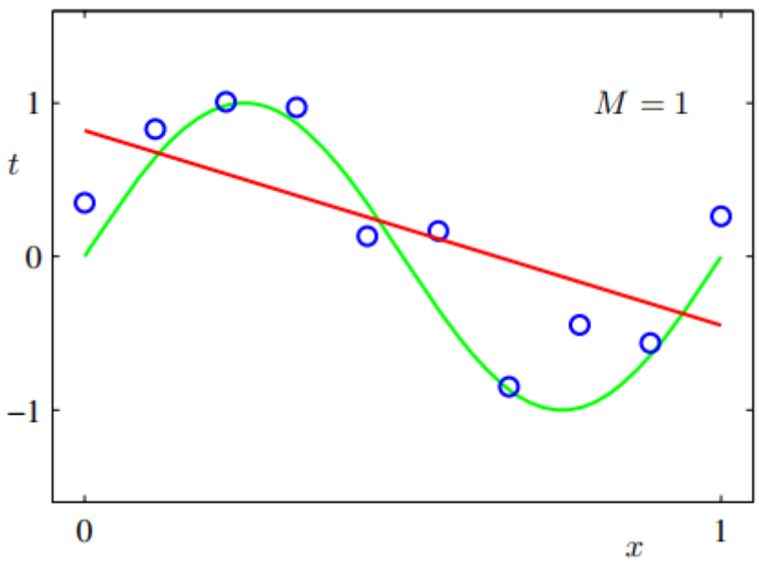
$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\
&= \frac{1}{2}\nabla_\theta \left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \\
&= \frac{1}{2}\nabla_\theta \operatorname{tr}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \\
&= \frac{1}{2}\nabla_\theta \left(\operatorname{tr}\theta^T X^T X\theta - 2\operatorname{tr}\vec{y}^T X\theta\right) \\
&= \frac{1}{2}\left(X^T X\theta + X^T X\theta - 2X^T\vec{y}\right) \\
&= X^T X\theta - X^T\vec{y}
\end{aligned}$$

Thus, the value of θ that minimizes J(θ) is given by

$$\theta = (X^T X)^{-1} X^T \vec{y}$$

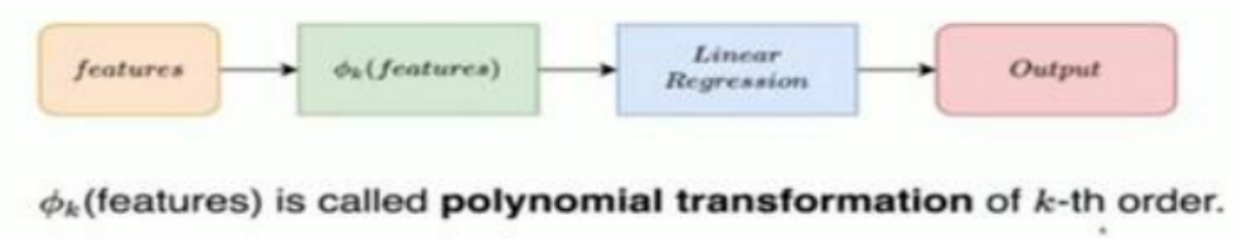
1.6 Polynomial Regression

- Many times the relationship between input features and output labels is non-linear.
- Simple linear models are unable to learn such mappings.



Clearly, this is not a great fit.

- We create polynomial features by combining existing input features.
- Then apply linear regression model on this representation.



ϕ_k(features) is called **polynomial transformation** of k-th order.

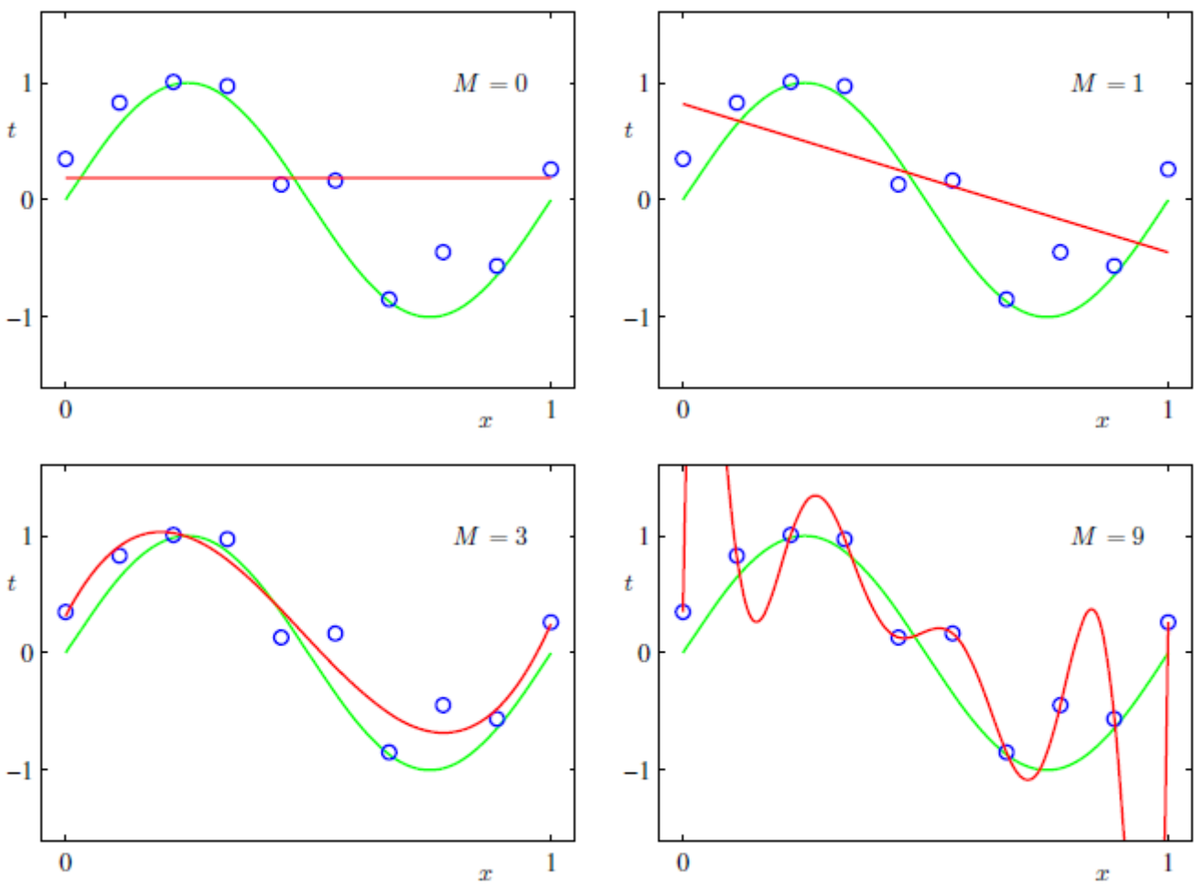
We define the error function as follows:

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2$$

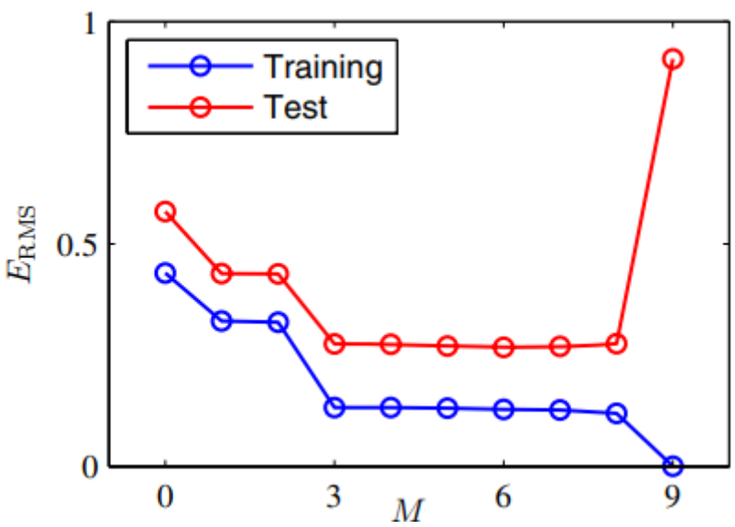
Here, x_n are the data points, t_n are the corresponding target values and $y(x_n,w)$ are the predicted values.

Model Selection

- Choosing the order, M of the polynomial.



- M = 0 and M = 1 give poor fits to the data and hence poor representations of the function.
- M = 3 seems to give the best fit to the function.
- For M = 9, we obtain an excellent fit to the training data and E(w*) = 0. The curve, however, oscillates wildly and gives a poor representation to the function. This is known as over-fitting.



- The goal is to achieve good generalization by making accurate predictions for new data.
- The test set error is a measure of how well we are doing in predicting the values of t for new data observations of x.
- Small values of M give relatively large values of the test set error.
- Values of M in the range $3 \leq M \leq 8$ give small values for the test set error, and these also give reasonable representations.

- For M = 9, the training set error goes to zero and the test set error has become very large.

The issue with Polynomial Regression

- Higher order polynomials are very flexible and more prone to overfitting.
- This issue is resolved using Regularization.

1.7 Regularization

- Controlling model complexity to fit complex models on relatively small dataset without overfitting.
- Idea is to add a penalty term in the loss function for the large weights.
- Penalty is a function of the weight vector.
- λ is a regularization rate that controls the amount of penalty to be added.

$$f(b) = E(b) + (\lambda/n)b^T b$$

- E(b) is the error function
- b is the weight vector
- b^T b is the penalty

Bias-Variance Tradeoff

- Low complexity models -> Low variance, High Bias
- High complexity models -> High variance, Low Bias

The approach to minimize E(b) unconditionally is to impose a restriction on the squared magnitude of the regression coefficients. Hence, we impose the following condition to b_1,...,b_p:

$$\sum_{j=1}^{p} b_j^2 \leq c$$

We can think of c as a budget for how much we can spend on the size of all the coefficients.

Now, we have to following constrained minimization:

$$\min_b \{\frac{1}{n}(Xb - y)^T(Xb - y)\} \; s.t. \; \|b\|_2^2 = b^T b \leq c$$

This can be rewritten as:

$$min_b\{E(b) + \frac{\lambda}{n}b^T b\}$$

Ridge Regression Estimate

$$
\begin{aligned}
f(\mathbf{b}) &= E(\mathbf{b}) + (\lambda/n)\mathbf{b}^\intercal\mathbf{b} \\
&= \frac{1}{n}(\mathbf{X}\mathbf{b}-\mathbf{y})^\intercal(\mathbf{X}\mathbf{b}-\mathbf{y}) + (\lambda/n)\mathbf{b}^\intercal\mathbf{b} \\
&= \frac{1}{n}\mathbf{b}^\intercal\mathbf{X}^\intercal\mathbf{X}\mathbf{b} - \frac{2}{n}\mathbf{b}^\intercal\mathbf{X}^\intercal\mathbf{y} + \frac{1}{n}\mathbf{y}^\intercal\mathbf{y} + \frac{\lambda}{n}\mathbf{b}^\intercal\mathbf{b}
\end{aligned}
$$

Deriving $f(\mathbf{b})$ with resepct to $\mathbf{b}$ we get:

$$
\nabla f(\mathbf{b}) = \frac{2}{n}\mathbf{X}^\intercal\mathbf{X}\mathbf{b} - \frac{2}{n}\mathbf{X}^\intercal\mathbf{y} + \frac{2\lambda}{n}\mathbf{b}
$$

Setting $\nabla f(\mathbf{b})$ equal to zero:

$$
\begin{aligned}
&\mathbf{X}^\intercal\mathbf{X}\mathbf{b} - \mathbf{X}^\intercal\mathbf{y} + \lambda\mathbf{b} = \mathbf{0} \\
&\implies \mathbf{b}(\mathbf{X}^\intercal\mathbf{X} + \lambda\mathbf{I}) = \mathbf{X}^\intercal\mathbf{y} \\
&\implies \mathbf{b} = (\mathbf{X}^\intercal\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\intercal\mathbf{y}
\end{aligned}
$$

Therefore, the solution is:

$$
\text{ridge coefficients:} \quad \mathbf{b}_{RR} = (\mathbf{X}^\intercal\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\intercal\mathbf{y}
$$

- λ = 0 causes the ridge solution to be the same as linear regression
- λ -> ∞ causes the ridge coefficients to collapse to zero.
- λ is a tuning parameter, i.e. it cannot be found analytically.

LASSO (Least Absolute Shrinkage and Selection Operator) Regression

- The mathematical setup is given as follows:

$$
\min_{\mathbf{b}\in\mathbb{R}^p}\left\{\frac{1}{n}\|\mathbf{y}-\mathbf{X}\mathbf{b}\|_2^2\right\} \quad \text{subject to} \quad \|\mathbf{b}\|_1 \le c
$$

$$
\|\mathbf{b}\|_1 \le c \iff \sum_{j=1}^{p}|b_j| \le c
$$

This is equivalent to:

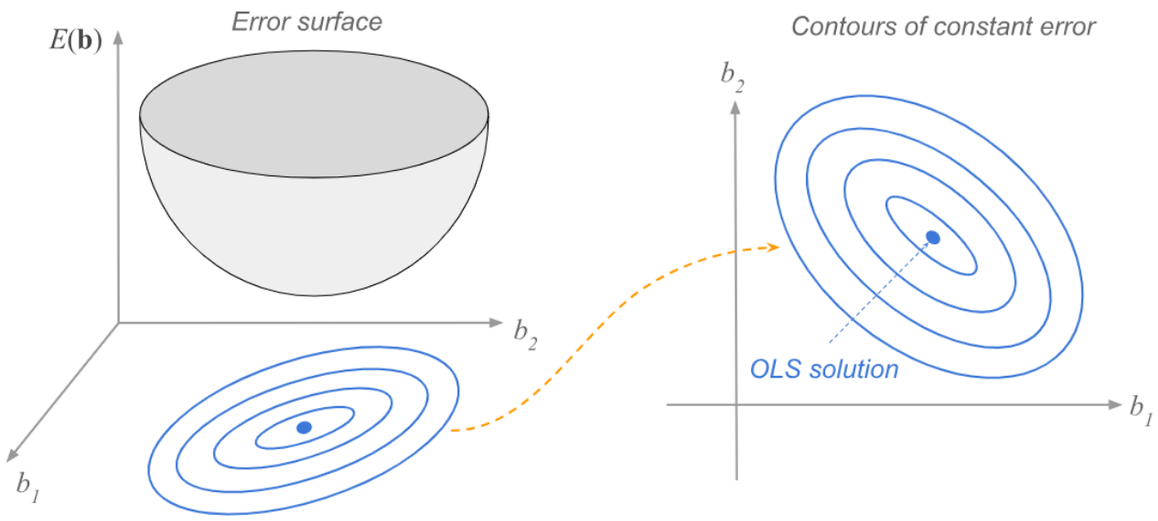
$$
\min_{\mathbf{b}}\left\{\frac{1}{n}(\mathbf{y}-\mathbf{X}\mathbf{b})^\intercal(\mathbf{y}-\mathbf{X}\mathbf{b}) + \lambda\sum_{j=1}^{p}|b_j|\right\}
$$

L1 & L2 norms

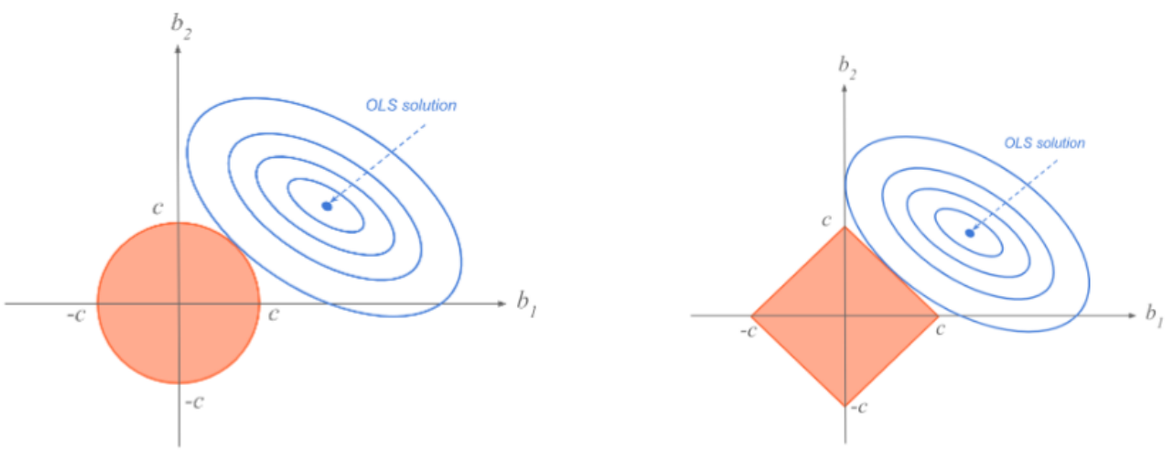
$$
\|w\|_1 = |w_1| + |w_2| + \cdots + |w_N|
$$

$$
\|w\|_2 = (|w_1|^2 + |w_2|^2 + \cdots + |w_N|^2)^{\frac{1}{2}}
$$

Error Surface



Consider two inputs x_1 and x_2 and corresponding parameters b_1 and b_2. The error function E(b) generates a bowl shaped (paraboloid) convex error surface. On the b_1, b_2 plane the locus of points such that $b^T b \leq c$ is a disk of radius c, centered at origin.

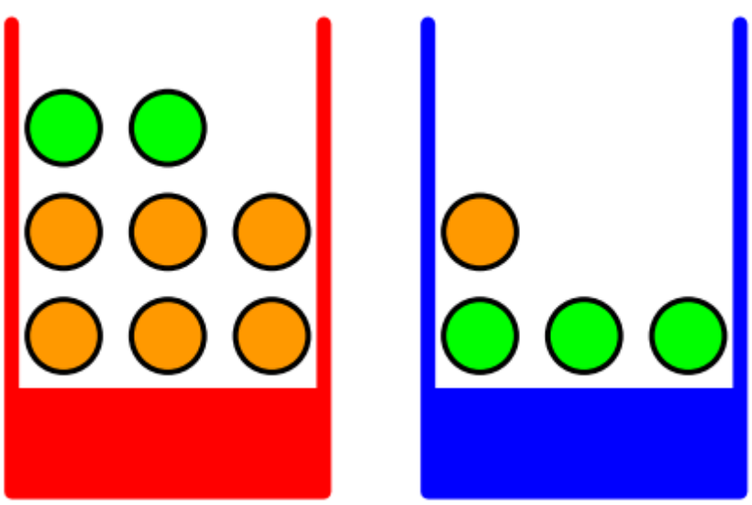


Ridge Regression LASSO Regression

1.8 Essentials of Probability Theory

- Provides a consistent framework for the quantification and manipulation of uncertainty and forms one of the central foundations for pattern recognition.

A simple example



- We have two boxes, one red and one blue, and in the red box we have 2 apples and 6 oranges, and in the blue box we have 3 apples and 1 orange.

- We randomly pick one of the boxes and from that box we randomly select an item of fruit, and having observed which sort of fruit it is we replace it in the box from which it came. We could imagine repeating this process many times.
- Let us suppose that in so doing we pick the red box 40% of the time and we pick the blue box 60% of the time, and that when we remove an item of fruit from a box we are equally likely to select any of the pieces of fruit in the box.
- The identity of the box that will be chosen is a random variable, which we shall denote by B.
- Similarly, the identity of the fruit is also a random variable and will be denoted by F.

Now we can compute the following probabilities as follows:

- p(B = r) = 4/10 ; probability of selecting the red box
- p(B = b) = 6/10 ; ; probability of selecting the blue box
- We can see that p(B = r) + p(B = b) = 1

Supposing we pick a box at random, and it turns out to be the blue box. Then the probability of selecting an apple in the blue box is given as p(F = a | B = b) = 3/4.

Similarly, we can write all the four conditional probabilities as follows:

$$\begin{aligned} p(F = a|B = r) &= 1/4 \\ p(F = o|B = r) &= 3/4 \\ p(F = a|B = b) &= 3/4 \\ p(F = o|B = b) &= 1/4. \end{aligned}$$

Using this, we now compute overall probability of choosing an apple as:

$$p(F = a) = p(F = a|B = r)p(B = r) + p(F = a|B = b)p(B = b) = \frac{1}{4}.\frac{4}{10} + \frac{3}{4}.\frac{6}{10} = \frac{11}{20}$$

from this it follows that:

$$p(F = o) = 1 - 11/20 = \frac{9}{20}$$

Now, suppose instead we are told that a piece of fruit has been selected and it is an orange, and we would like to know which box it came from. This requires that we evaluate the probability distribution over boxes conditioned on the identity of the fruit. To solve this, we use Bayes' Theorem as follows:

$$p(B = r|F = o) = \frac{p(F = o|B = r)p(B = r)}{p(F = o)} = \frac{3}{4}.\frac{4}{10}.\frac{20}{9} = \frac{2}{3}$$

and it clearly follows that:

$$p(B = b|F = o) = 1 - \frac{2}{3} = \frac{1}{3}$$

- p(B) is called the **prior probability** because it is the probability available before we observe the identity of the fruit.
- p(B|F) is called the posterior probability because it is the probability obtained after we have observed F.

A summary of the basic rules

1. Sum Rule:

$$p(X) = \sum_{Y} p(X, Y)$$

2. Product Rule:

$$p(X, Y) = \frac{p(Y|X)p(Y)}{p(X)}$$

3. Bayes' Theorem:

$$p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$$

- If the joint distribution of two variables factorizes into the product of the marginals, so that p(X,Y) = p(X)p(Y), then X and Y are said to be independent.

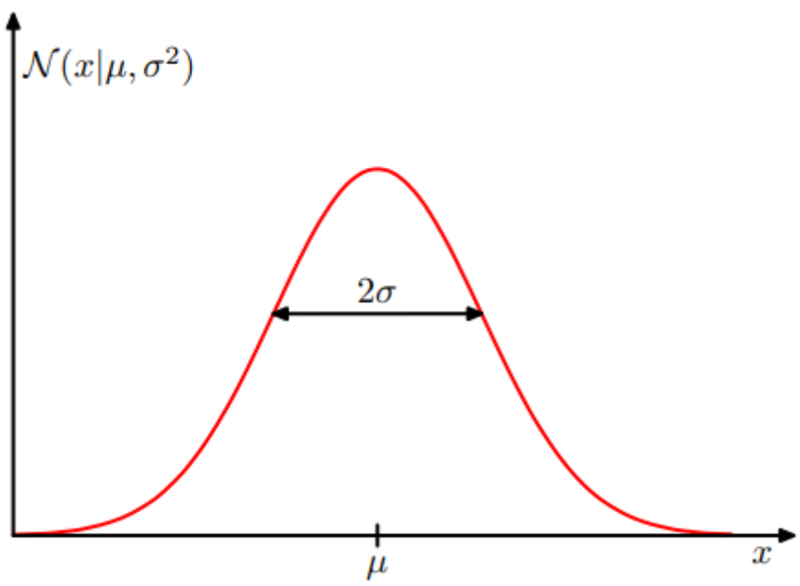
Gaussian/Normal Distribution

For a single real-valued variable x, we define the Gaussian distribution as

$$\mathcal{N}\left(x|\mu, \sigma^2\right) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

where μ is the mean, σ^2 is the variance and σ is the standard deviation. We also define precision as the reciprocal of the variance, i.e. $\beta = 1/\sigma^2$.

The univariate Gaussian distribution:



- Suppose that we have a data set of observations $x = (x_1,...,x_N)^T$, representing N observations of the scalar variable x.
- The observations are drawn independently from a Gaussian distribution whose mean μ and variance σ^2 are unknown, and we would like to determine these parameters from the data set.
- Data points that are drawn independently from the same distribution are said to be independent and identically distributed, which is often abbreviated to i.i.d.

We can write the probability of the dataset as:

$$p(\mathbf{x}|\mu,\sigma^2) = \prod_{n=1}^{N} \mathcal{N}\left(x_n|\mu,\sigma^2\right).$$

This is the likelihood function of the Gaussian.

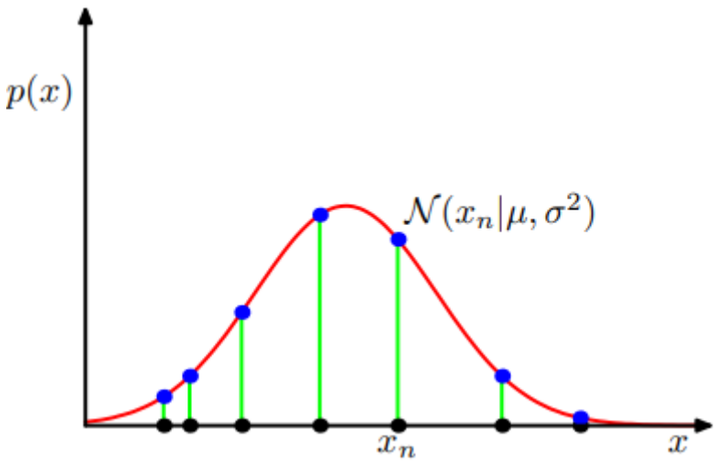
We denote it as

$$L(\mu,\sigma;data) = P(data;\mu,\sigma)$$

LHS: The likelihood of the parameters μ and σ taking certain values given that data is observed.
RHS: Probability density of observing the data with model parameters μ and σ.

1.9 Maximum Likelihood Estimation

- We want to know the values of μ and σ that result in the curve that best fits the data.



In practice, we observe that maximizing the log of the likelihood function is easier. Hence, we maximize

$$\ln p\left(\mathbf{x}|\mu,\sigma^2\right) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 - \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln(2\pi).$$

and obtain

$$\mu_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N} x_n \qquad \sigma^2_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N}(x_n-\mu_{\mathrm{ML}})^2$$

which is nothing but the sample mean and sample variance respectively.

Derivation of the result

Our first step will be to use the log product rule $\log_b(MN) = \log_b(M) + \log_b(N)$ for $b > 0$

$$\begin{aligned}\mathcal{LL} &= \sum_{n=1}^{N} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp^{-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)}\right) \\ &= \sum_{n=1}^{N} \left(\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(\exp^{-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)}\right)\right)\end{aligned}$$

Now we will use the log quotient rule: $\log_b\left(\frac{M}{N}\right) = \log_b(M) - \log_b(N)$.

$$\begin{aligned}\mathcal{LL} &= \sum_{n=1}^{N} \left(\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \log\left(\exp^{-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)}\right)\right) \\ &= \sum_{n=1}^{N} \left(\log(1) - \log\left(\sqrt{2\pi\sigma^2}\right) + \log\left(\exp^{-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)}\right)\right)\end{aligned}$$

Now, we'll use the log power rule: $\log_b(M^n) = n\log_b M$

$$\begin{aligned}\mathcal{LL} &= \sum_{n=1}^{N} \left(\log(1) - \log\left(\sqrt{2\pi\sigma^2}\right) + \log\left(\exp^{-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)}\right)\right) \\ &= \sum_{n=1}^{N} \left(\log(1) - \log\left(\sqrt{2\pi\sigma^2}\right) + \left(-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right) \cdot \log(e)\right)\right)\end{aligned}$$

$$\begin{aligned}\mathcal{LL} &= \sum_{n=1}^{N} \left(\log(1) - \log\left(\sqrt{2\pi\sigma^2}\right) + \left(-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right) \cdot \log(e)\right)\right) \\ &= \sum_{n=1}^{N} \left(-\log\left(\sqrt{2\pi\sigma^2}\right) + \left(-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)\right)\right)\end{aligned}$$

We can apply the power rule one more time (remember that $\sqrt{x} = x^{1/2}$).

$$\begin{aligned}\mathcal{LL} &= \sum_{n=1}^{N} \left(-\log\left(\sqrt{2\pi\sigma^2}\right) + \left(-\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)\right)\right) \\ &= \sum_{n=1}^{N} \left(-\frac{1}{2} \cdot \log(2\pi\sigma^2) - \frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)\right)\end{aligned}$$

Now for some basic algebra simplification:

$$\begin{aligned}\mathcal{LL} &= \sum_{n=1}^{N} \left(-\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right)\right) \\ &= -\frac{N}{2}\log(2\pi\sigma^2) + \sum_{n=1}^{N} -\frac{1}{2}\left(\frac{(x_n-\mu)^2}{\sigma^2}\right) \\ &= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2\end{aligned}$$

1.10 Probabilistic Interpretation of Regression

We assume that the target variables and inputs are related as

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

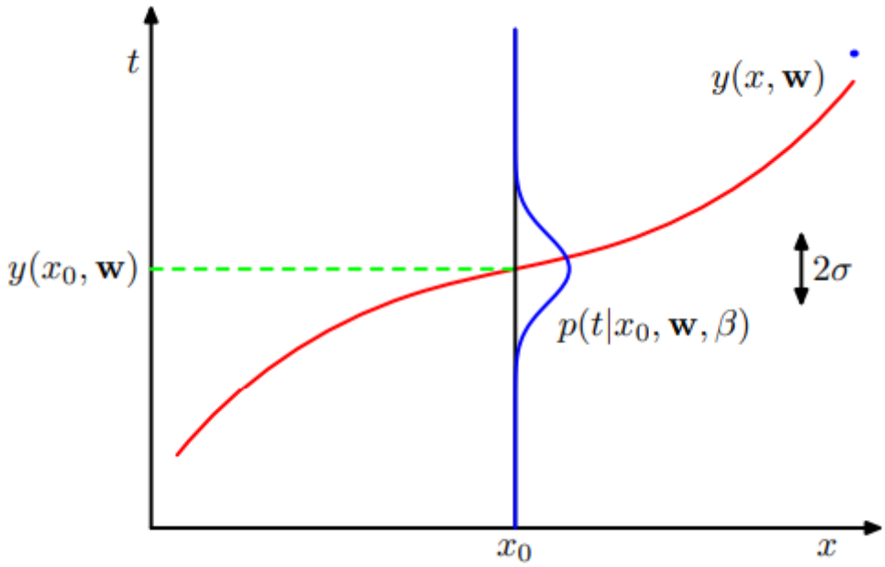
- $\varepsilon^{(i)}$ is an error term that captures either unmodeled effects or random noise.
- We also assume that $\varepsilon^{(i)}$ are i.i.d according to a Gaussian distribution with $\mu = 0$ and σ^2.

The density of $\varepsilon^{(i)}$ is given by

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

and it implies that

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}\right)$$



- We can express our uncertainty over the value of the target variable using a probability distribution.
- For this purpose, we shall assume that, given the value of x, the corresponding value of t has a Gaussian distribution with a mean equal to the value y(x, w) of the polynomial curve.
- Given X (the design matrix, containing all $x^{(i)}$'s) and θ, we wish to know the distribution of $y^{(i)}$'s.

The probability of the data is given by

$$p(\vec{y}|X;\theta)$$

Since we want to view it as a function of θ, we define it as the likelihood function

$$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X;\theta)$$

This can be re-written as follows

$$L(\theta) = \prod_{i=1}^{n} p(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2})$$

Now, given this probabilistic model, we want to choose our best guess of the parameters θ. The principal of maximum likelihood says that we should choose θ so as to make the data as high probability as possible. i.e. we should choose θ to maximize L(θ).

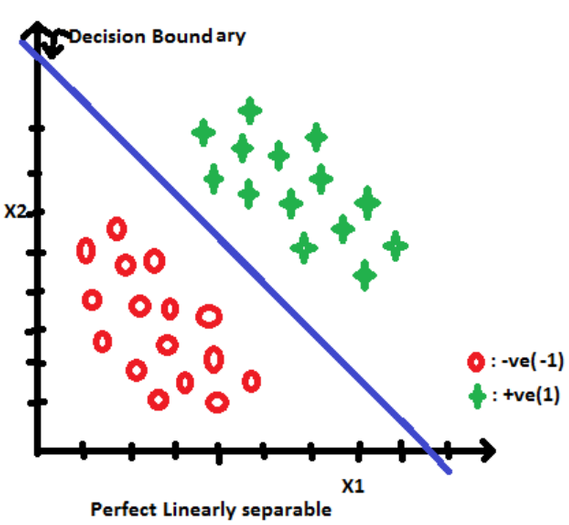
To make our computations simpler, we choose to maximize the log likelihood instead:

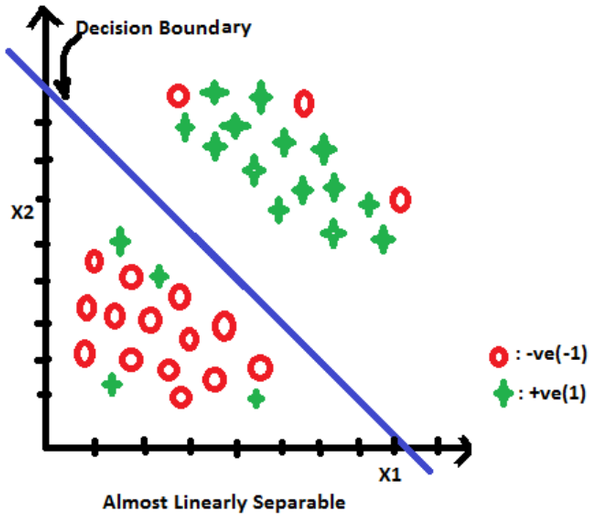
$$\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \log \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2
\end{aligned}$$

Chapter-2 Linear Models for Classification

2.1 The Classification Problem

- **Goal:** Take an input vector x and assign it to one of K discrete classes C_k where k = 1,...,K.
- The classes are usually taken as disjoint, so each input is assigned to only one class.
- The input space is thus divided into decision regions whose boundaries are called decision boundaries or decision surfaces.
- The decision surfaces are linear functions of the input vector x and defined by (D-1) - dimensional hyperplanes within the D-dimensional input space.
- The datasets whose classes can be separated exactly by linear decision surfaces are said to be linearly separable.

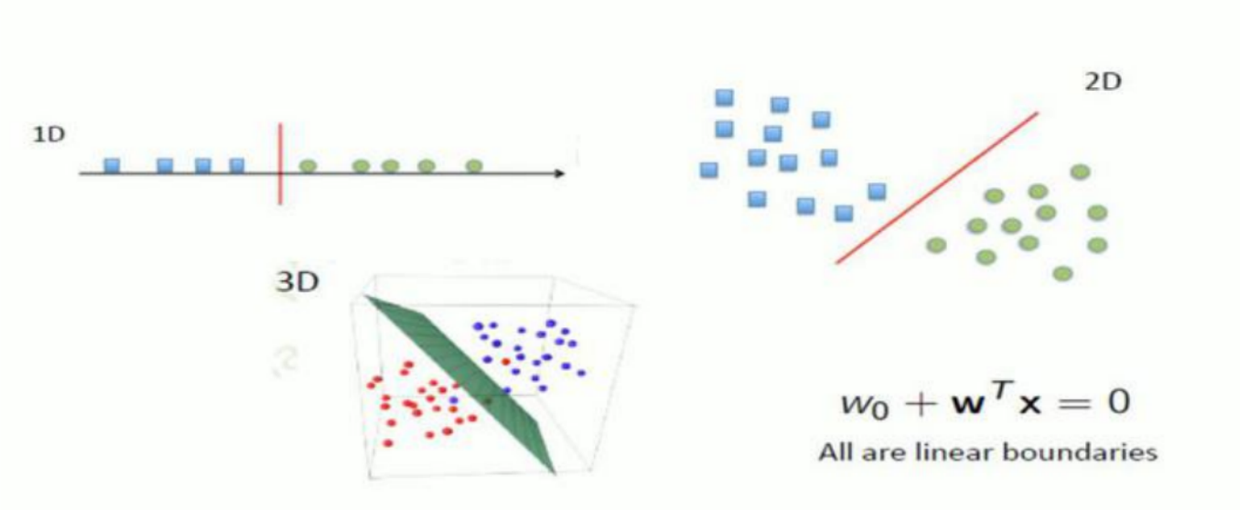




- In the case of two-class problems, we use the binary representation in which there is a single target variable t ∈ {0, 1} such that t = 1 represents class C_1 and t = 0 represents class C_2.
- We can interpret the value of t as the probability that the class is C_1, with the values of probability taking only the extreme values of 0 and 1.
- For K > 2 classes, it is convenient to use a 1-of-K coding scheme in which t is a vector of length K such that if the class is C_j , then all elements t_k of t are zero except element t_j, which takes the value 1.
- For instance, if we have K = 5 classes, then a pattern from class 2 would be given the target vector $t = (0, 1, 0, 0, 0)^T$.
- Again, we can interpret the value of t_k as the probability that the class is C_k.
- For classification problems, we wish to predict discrete class labels, or more generally posterior probabilities that lie in the range (0, 1). To achieve this, we consider a generalization of this model in which we transform the linear function of w using a nonlinear function f(·) so that

$$y(x) = f(w^T x + w_0)$$

This is known as an activation function or a link function.

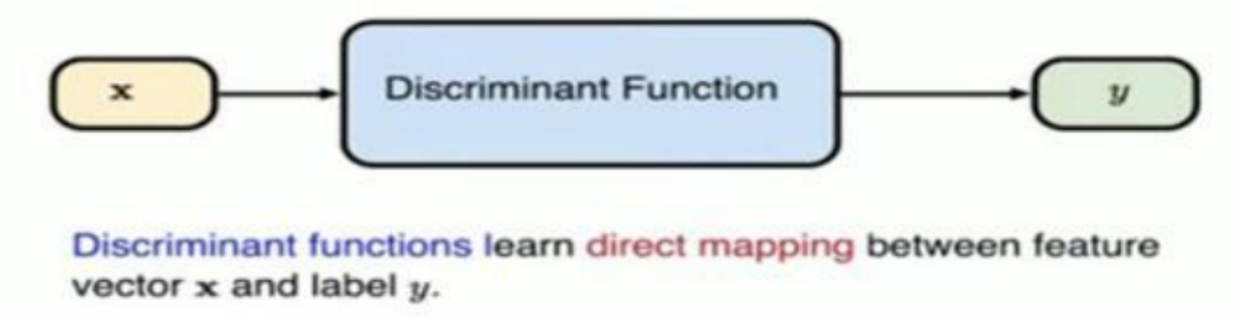


- The decision surfaces correspond to y(x) = constant, so that $w^Tx + w_0$ = constant and hence the decision surfaces are linear functions of x, even if the function f(·) is nonlinear.
- For this reason, this class of models are called generalized linear models.

Approaches to the classification problem

1. Constructing a discriminant function that directly assigns each vector x to a specific class.
2. Model the conditional probabilities in an inference stage, and then use it to make decisions.
    1. Model them directly by representing them as parametric models
    2. Generative approach, i.e. model the class-conditional densities with prior probabilities

2.2. Discriminant Functions



It takes an input vector x and assigns it to one of the K classes, denoted C_k

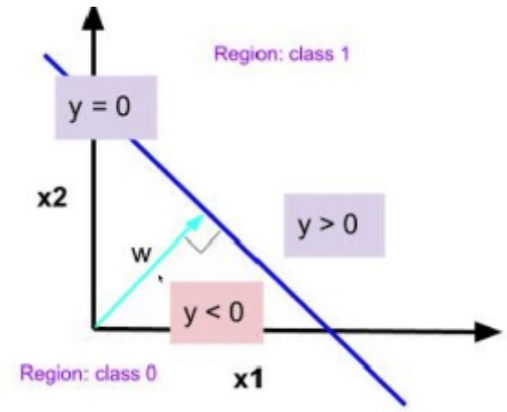
For Two Classes (K=2)

The simplest representation can be given as

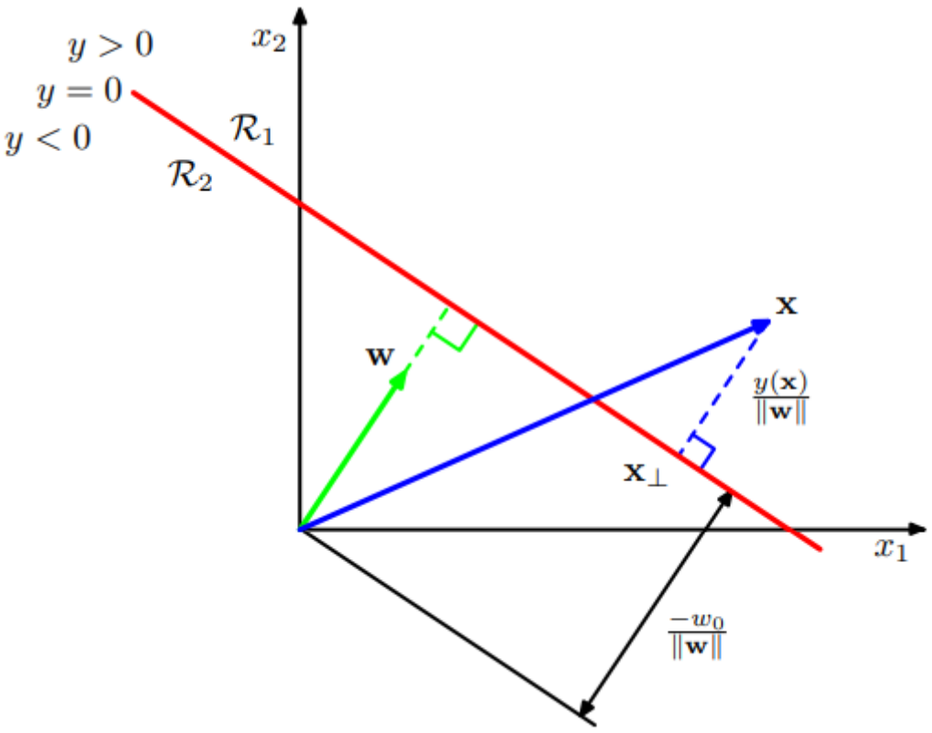
$$y(x) = w^T x + w_0$$

where w is the weight vector and w_0 is a bias.

- An input vector x is assigned to class C_1 if y(x) ≥ 0 and to class C_2 otherwise.
- The corresponding decision boundary is defined by the relation y(x) = 0, which corresponds to a (D-1) - dimensional hyperplane withing the D-dimensional input space.



- If we consider two points x_A and x_B lying on the decision surface, because $y(x_A) = y(x_B) = 0$, we get $w^T(x_A - x_B) = 0$.
- Hence, vector w is orthogonal to every vector lying within the decision surface, and so w determines the orientation of the surface.



- Similarly, if x is a point on the decision surface, then y(x) = 0, and the normal distance from the origin to the decision surface is given by

$$\frac{w^T x}{\|w\|} = -\frac{w_0}{\|w\|}$$

- y(x) gives the signed measure of the perpendicular distance r of point x from decision surface.
- Consider an arbitrary point x and let $x_\perp$ be its orthogonal projection onto the decision surface, so that

$$x = x_\perp + r\frac{w}{\|w\|}$$

- Also we know

$$w^T x + w_0 = y(x)$$

and

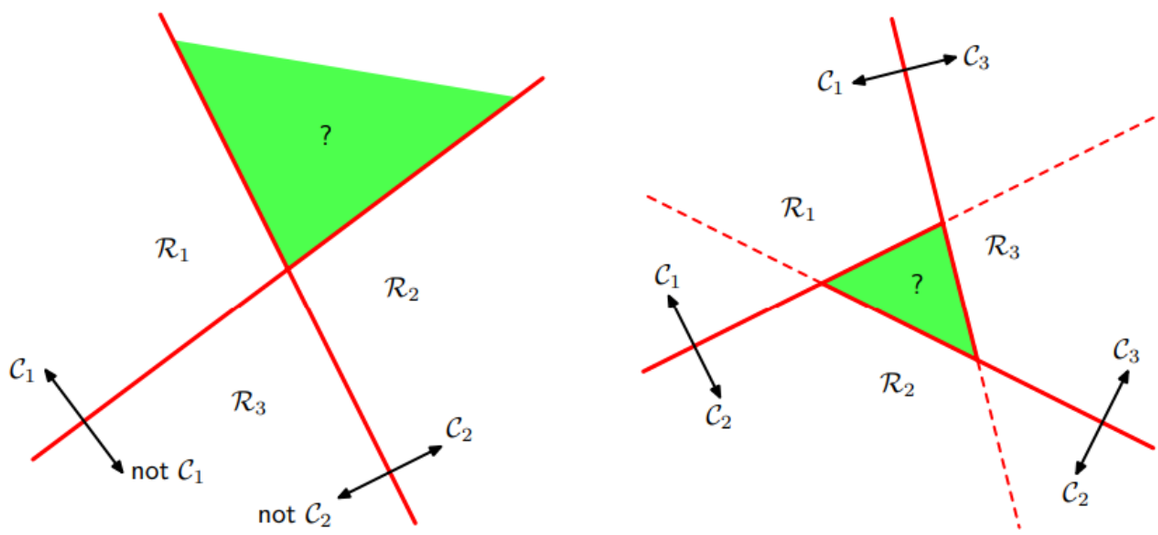
$$w^T x_\perp + w_0 = y(x_\perp) = 0$$

- Finally, we obtain

$$r = \frac{y(x)}{\|w\|}$$

For Multiple Classes (K>2)

1. One vs Rest - build K-1 discriminant functions. Each solves a two class classification.
2. One vs One - one discriminant function per pair of classes. K(K-1)/2 functions.



These approaches lead to the problem of ambiguous regions.

To avoid this we consider a single K-class discriminant comprising K linear functions for the form

$$y_k(w) = w_k^T x + w_{k_0}$$

Then we assign a point x to class C_k if $y_k(x) > y_j(x)$ for all $j \neq k$

The decision boundary between class C_k and class C_j is therefore given by $y_k(x) = y_j(x)$ and hence corresponds to a (D − 1)-dimensional hyperplane defined by

$$(w_k - w_j)^T x + (w_{k_0} - w_{j_0}) = 0$$

2.3 Least Squares Classification

- Consider a K-class problem with 1-of-K binary coding scheme for target vector t.
- Each class C_k has its own linear model

$$y_k(x) = w_k^T x + w_{k_0}$$

- This can be rewritten as

$$\tilde{W}^T \tilde{x}$$

$$\textit{where } \tilde{W} \textit{ is a matrix whose } k^{th} \textit{ column comprises the } D+1-\textit{dimensional vector } \tilde{w}_k = (w_{k_0}, w_k^T)^T$$

$$\textit{and } \tilde{x} \textit{ is the corresponding augmented input vector } (1, x^T)^T \textit{ with dummy input } x_0 = 1$$

- $$\tilde{W} = \begin{bmatrix} w_1^0 & \cdots & w_k^0 \\ \vdots & \ddots & \vdots \\ w_1^D & \cdots & w_k^D \end{bmatrix}$$

- A new input x is assigned to the class for which the output y_k is largest.
- Now we determine the parameter matrix by minimizing a sum of squares error function.

- Consider a training data set $\{x_n, t_n\}$ where n = 1,...,N and define matrix T whose n^{th} row is the vector t_n^T together with matrix X whose n^{th} row to x_n^T. The error function can be written as

$$E_D(\tilde{W}) = \frac{1}{2} Tr(\tilde{X}\tilde{W} - T)^T(\tilde{X}\tilde{W} - T)$$

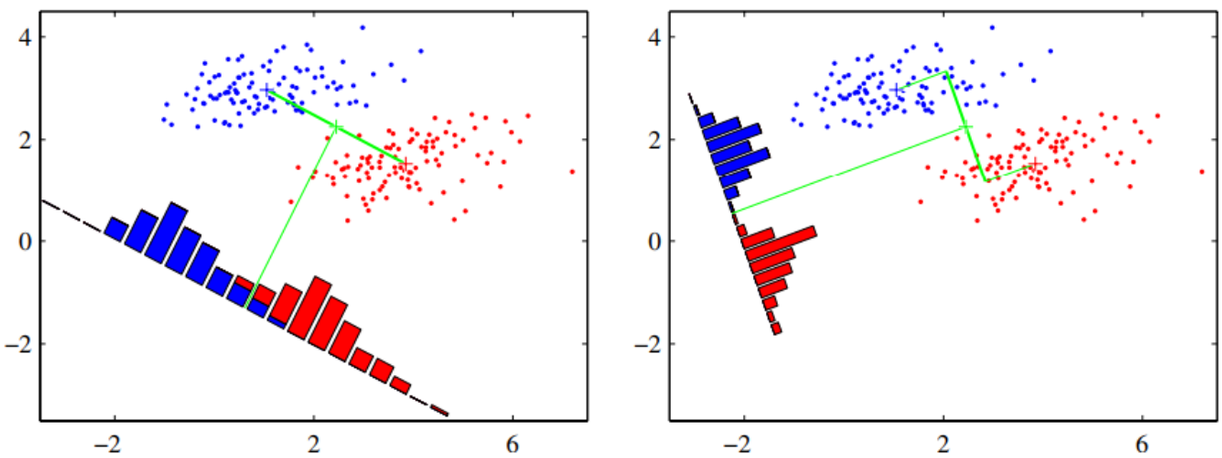
- Setting the derivative to zero and rearranging we get

$$\tilde{W} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T T = \tilde{X}^\dagger T$$

- Here, X† is the pseudo-inverse of the matrix X
- The least squares solutions lack robustness to outliers and can give poor results.

2.4 Fisher's Linear Discriminant

- Take a D-dimensional input vector and project it down to 1-dimension using $y = w^T x$
- **Idea:** Find the projection that maximizes the class separation.



- We consider a 2-class problem with N_1 points of class C_1 and N_2 points of class C_2, so that the mean vectors are given as

$$m_1 = \frac{1}{N_1}\sum_{n\in C_1} x_n \qquad m_2 = \frac{1}{N_2}\sum_{n\in C_2} x_n$$

- The simplest measure of separation of classes when projected onto w, is the separation of project class means. That is

$$Choose\ w\ to\ maximize\ m_2 - m_1 = w^T(m_2 - m_1)$$

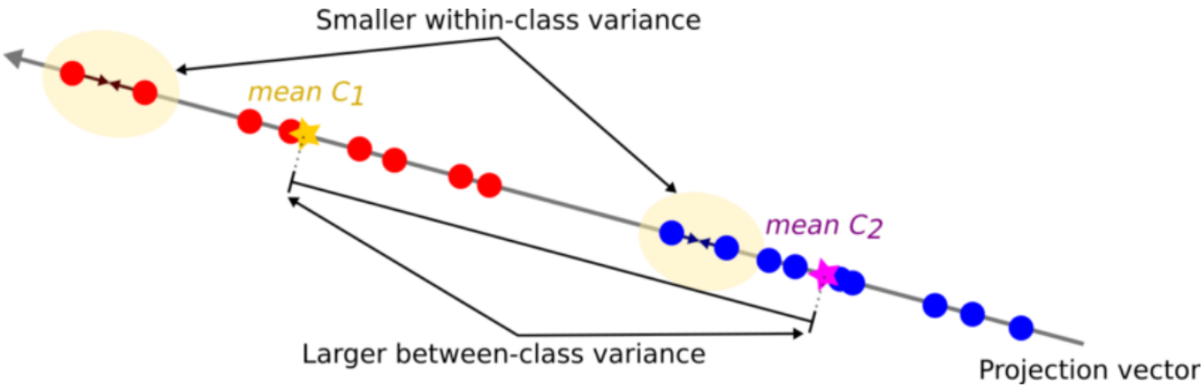
- where $m_k = w^T m_k$ is the mean of projected data from C_k
- The project formula transforms the set of labelled data points in x into a labelled set of 1-dimensional space y.
- The within class variance of transformed data from C_k is

$$s_k^2 = \sum_{n\in C_k}(y_n - m_k)^2$$

- where $y_n = w^T x_n$
- The total within class variance for the whole dataset can be defined to be

$$s_1^2 + s_2^2$$

Fisher's Criterion



- The ratio of between class variance to the within class variance, i.e.

$$J(w) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

- We can rewrite the criterion

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

where S_B is the between class covariance matrix given by

$$S_B = (m_2 - m_1)(m_2 - m_1)^T$$

and S_W is the total within class covariance matrix give by

$$S_W = \sum_{n \in C_1} (x_n - m_1)(x_n - m_1)^T + \sum_{n \in C_2} (x_n - m_2)(x_n - m_2)^T$$

- Differentiating with respect to w, we see that J(w) is maximized when

$$(w^T S_B w) S_W w = (w^T S_W w) S_B w$$

- $S_B w$ is always in the direction of (m_2 - m_1) and since we do not care about the magnitude of w, the scalar factors can be dropped and after multiplying both sides by S_W^{-1} we obtain

$$w \propto S_W^{-1}(m_2 - m1)$$

- This result is known as Fisher's Linear Discriminant.
- Although strictly it is not a discriminant but rather a specific choice of direction for projection of the data down to one dimension.
- However, the projected data can subsequently be used to construct a discriminant, by choosing a threshold y_0 so that we classify a new point as belonging to C_1 if $y(x) \geq y_0$ and classify it as belonging to C_2 otherwise.

2.5 Perceptron

- Another example of a linear discriminant model is the perceptron of Rosenblatt (1962), which occupies an important place in the history of pattern recognition algorithms.
- It corresponds to a two-class model in which the input vector x is first transformed using a fixed nonlinear transformation to give a feature vector φ(x), and this is then used to construct a generalized linear model of the form

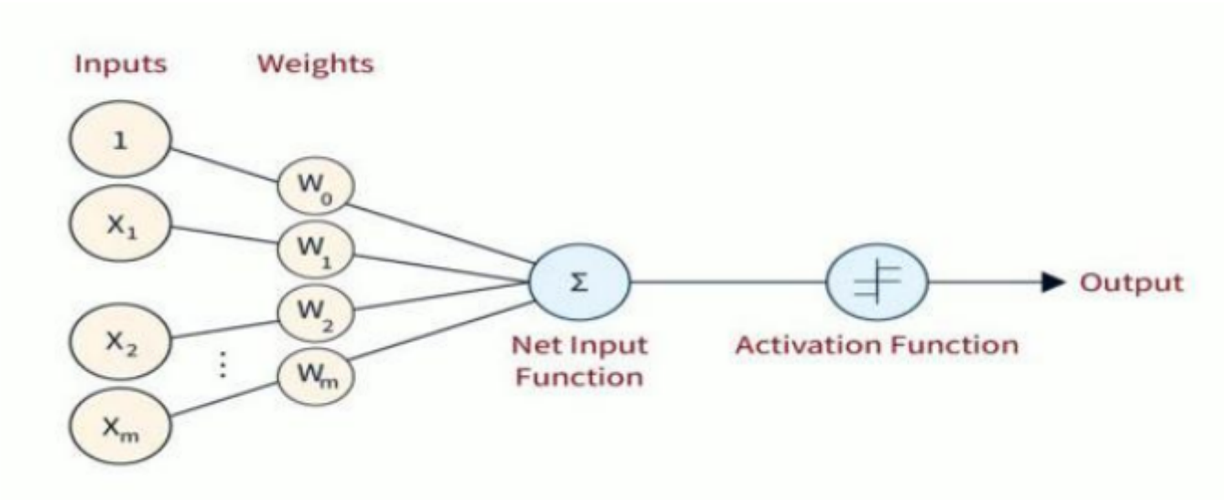
$$y(x) = f(w^T \phi(x))$$

- The non-linear activation function f(.) is given by a step function for the form

$$f(a) = \begin{cases} +1 & , a \geq 0 \\ -1 & , a < 0 \end{cases}$$

- For the perceptron, it is more convenient to use target values t = +1 for class C_1 and t = -1 for class C_2, which matches the choice of the activation function.

Perceptron Learning Algorithm



- The error function is known as perceptron criterion.
- We are seeking a weight vector w such that patterns x_n in class C_1 will have $w^T\varphi(x_n) > 0$, whereas patterns x_n in class C_2 have $w^T\varphi(x_n) < 0$.
- Using the $t \in \{-1, +1\}$ target coding scheme it follows that we would like all patterns to satisfy $w^T\varphi(x_n)t_n > 0$.
- The perceptron criterion associates zero error with any pattern that is correctly classified, whereas for a misclassified pattern x_n it tries to minimize the quantity $-w^T\varphi(x_n)t_n$.
- The perceptron criterion is therefore given by

$$E_p(w) = -\sum_{n \in M} w^T \phi_n t_n$$

  where M denotes the set of all misclassified patterns.
- The contribution to the error associated with a particular misclassified pattern is a linear function of w in regions of w space where the pattern is misclassified and zero in regions where it is correctly classified.
- The total error function is therefore piecewise linear.

Optimization

1. Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
2. For each training example $(\mathbf{x}^{(i)}, y^{(i)})$:

$$\hat{y}^{(i)} = \text{sign}\left(\mathbf{w}^T \phi(\mathbf{x}^{(i)})\right)$$

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha\left(y^{(i)} - \hat{y}^{(i)}\right)\phi(\mathbf{x}^{(i)})$$

- The perceptron learning algorithm has a simple interpretation, as follows.
- We cycle through the training patterns in turn, and for each pattern x_n we evaluate the perceptron function. If the pattern is correctly classified, then the weight vector remains

unchanged, whereas if it is incorrectly classified, then for class C_1 we add the vector $\varphi(x_n)$ onto the current estimate of weight vector w while for class C_2 we subtract the vector $\varphi(x_n)$ from w.

2.6 Probabilistic Generative Models

- Model class conditional densities p(x | C_k) as well as priors p(C_k) and then computer posterior probabilities p(C_k | x) through Bayes' Theorem.
- For the case of 2 classes, the probability for class C_1 can be written as

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)} = \frac{1}{1+\exp(-a)} = \sigma(a)$$

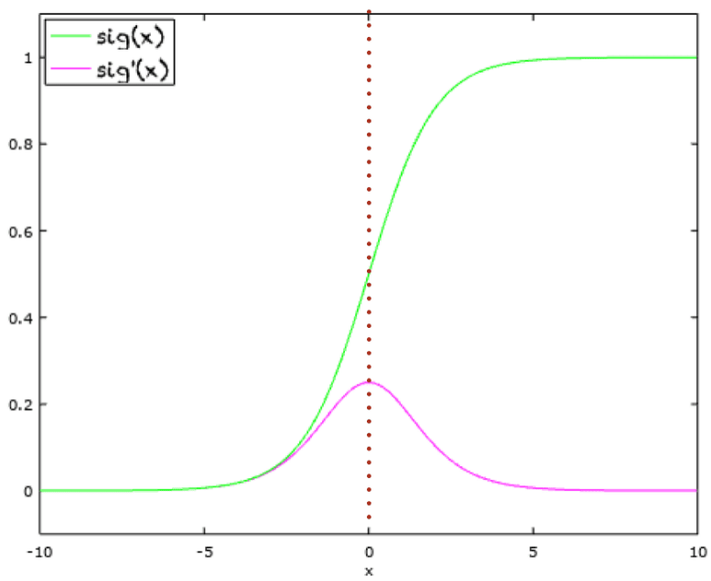
where

$$a = \ln \frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$$

- σ(a) is termed as the logistic sigmoid function. The term sigmoid means 'S-shaped'.
- This is sometimes also called squashing function as it maps the whole real axis into a finite interval.
- It maps the feature space into a probability function.

Properties of the sigmoid function

1. Symmetry: σ(−a)=1 − σ(a)
2. Inverse: a = ln (σ/(1-σ)) known as the logit function. Represents log of the ratio of probabilities.



Plot of σ(x) and its derivate σ′(x)

Domain: (−∞, +∞)
Range: (0, +1)
σ(0) = 0.5

Other properties

$$\sigma(x) = 1 - \sigma(-x)$$

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

- Log Odds: ln(p(C_1 | x)/p(C_2 | x)) for two classes
- For probability p, sigmoid(logit(p)) = p
- The logit function is given as

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

- Logistic function is the inverse of logit (maps values from the range (-∞,+∞) into [0,1])

$$
\begin{aligned}
logit(P(x)) &= \log(\frac{P(x)}{1-P(x)}) \\
X &= \log(\frac{P(x)}{1-P(x)}) && (X \in (-\infty, +\infty)) \\
e^X &= \frac{P(x)}{1-P(x)} && (e^{\log(x)} = x) \\
\frac{1}{e^X} &= \frac{1-P(x)}{P(x)} && (\text{invert fractions}) \\
\frac{1}{e^X} &= \frac{1}{P(x)} - 1 && (\frac{1-P(x)}{P(x)} = \frac{1}{P(x)} - \frac{P(x)}{P(x)}) \\
\frac{1}{e^X} + 1 &= \frac{1}{P(x)} \\
\frac{1+e^X}{e^X} &= \frac{1}{P(x)} \\
P(x) &= \frac{e^X}{1+e^X} \overset{(*)}{=} \frac{1}{1+e^{-X}} && (*\text{multiply by } \frac{e^{-X}}{e^{-X}})
\end{aligned}
$$

- For K>2, we define the normalized exponential and it can be regarded as a multiclass generalization of the logistic sigmoid.

$$
\begin{aligned}
p(\mathcal{C}_k|\mathbf{x}) &= \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} \\
&= \frac{\exp(a_k)}{\sum_j \exp(a_j)}
\end{aligned}
$$

$$
a_k = \ln p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k).
$$

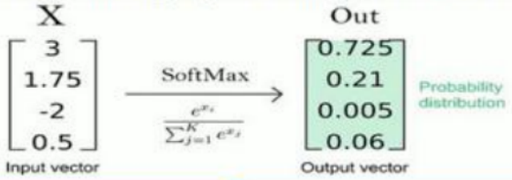
- This is also known as the SoftMax function. It turns logits into probabilities by taking exponential of each output and then normalizing each number by the sum of those exponents so that the entire output vector adds up to one.

$$
X = \begin{bmatrix} logit(P(Y = class1|x)) \\ logit(P(Y = class2|x)) \\ \vdots \\ logit(P(Y = classk|x)) \end{bmatrix}
$$

To convert X into a probability distribution we can apply the exponential function and obtain the odds ∈ [0,+∞)

$$
e^X = e^{\log(odds)} = odds
$$

Finally, we can just normalize the result by dividing by the sum of all the odds, so that the range value changes from [0,+∞) to [0,1] and we make sure that the sum of all the elements is equal to 1, thus building a probability distribution over all the predicted classes.



Derivative of the sigmoid

The derivative of the sigmoid is $\frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$.

Here's a detailed derivation:

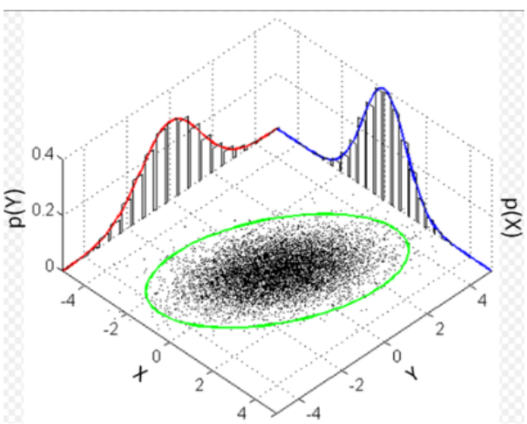
$$\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\
&= \frac{d}{dx}\left(1+\mathrm{e}^{-x}\right)^{-1} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x) \cdot (1-\sigma(x))
\end{aligned}$$

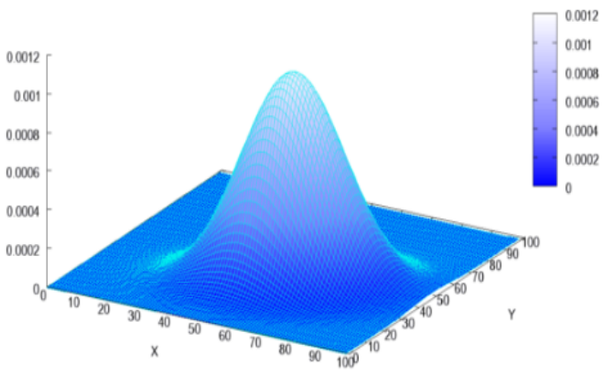
Continuous Inputs

- Let us assume that the class-conditional densities are Gaussian and then explore the resulting form for the posterior probabilities. To start with, we shall assume that all classes share the same covariance matrix. Thus the density for class C_k is given by

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^{\mathrm{T}}\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right\}.$$

- For a D-dimensional vector x, the multivariate Gaussian distribution takes the above form, where μ is a D-dimensional mean vector, Σ is a DxD covariance matrix, and |Σ| denotes the determinant.





- We consider the case of 2 classes

$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0)$$

where we have defined

$$\begin{aligned} \mathbf{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ w_0 &= -\frac{1}{2}\boldsymbol{\mu}_1^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}. \end{aligned}$$

$$\begin{aligned} a &= \ln\frac{p(\mathbf{x}|C_1)p(C_1)}{p(\mathbf{x}|C_2)p(C_2)} = \ln\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1,\boldsymbol{\Sigma}) - \ln\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2,\boldsymbol{\Sigma}) + \ln\frac{p(C_1)}{p(C_2)} \\ &= -\frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_1) + \frac{1}{2}\ln|\boldsymbol{\Sigma}| + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_2)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_2) + \ln\frac{p(C_1)}{p(C_2)} \end{aligned}$$

Maximum Likelihood Solution

- Once we have specified a parametric functional form for the class-conditional densities $p(x|C_k)$, we can then determine the values of the parameters, together with the prior class probabilities $p(C_k)$, using maximum likelihood.
- This requires a data set comprising observations of x along with their corresponding class labels.
- We consider the case of two classes, each having a Gaussian class-conditional density with a shared covariance matrix, and suppose we have a data set $\{x_n, t_n\}$ where n = 1,...,N. Here t_n = 1 denotes class C_1 and t_n = 0 denotes class C_2. We denote the prior class probability $p(C_1) = \pi$, so that $p(C_2)=1 - \pi$. For a data point x_n from class C_1, we have t_n = 1 and hence

$$p(x_n, C_1) = p(C_1)p(x_n|C_1) = \pi\mathcal{N}(x_n|\mu_2, \Sigma)$$

- Similarly for class C_2, we have t_n = 0 and hence

$$p(x_n, C_2) = p(C_2)p(x_n|C_2) = (1-\pi)\mathcal{N}(x_n|\mu_2, \Sigma)$$

- Thus the likelihood function is given by

$$p(t, X|\pi, \mu_1, \mu_2, \Sigma) = \prod_{n=1}^{N} [\pi\mathcal{N}(x_n|\mu_1\Sigma)]^{t_n}[(1-\pi)\mathcal{N}(x_n|\mu_2, \Sigma)]^{1-t_n}$$

- The log likelihood is given by

$$\ln p(t, X|\pi, \mu_1, \mu_2, \Sigma) = \sum_{n=1}^{N} t_n \ln\pi + t_n \ln\mathcal{N}(x_n|\mu_1, \Sigma) + (1-t_n)\ln(1-\pi) + (1-t_n)\ln\mathcal{N}(x_n|\mu_2, \Sigma$$

- Maximizing with respect to π we get

$$\pi = \frac{1}{N}\sum_{n=1}^{N} t_n = \frac{N_1}{N_1 + N_2}$$

- Maximizing with respect to μ_1 we get

$$\mu_1 = \frac{1}{N_1}\sum_{n=1}^{N} t_n x_n$$

- Maximizing with respect to μ_2 we get

$$\mu_2 = \frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)x_n$$

- Maximizing with respect to Σ we get

$$-\frac{1}{2}\sum_{n=1}^{N} t_n \ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_{n=1}^{N} t_n(\mathbf{x}_n-\boldsymbol{\mu}_1)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_1)$$

$$-\frac{1}{2}\sum_{n=1}^{N}(1-t_n)\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_{n=1}^{N}(1-t_n)(\mathbf{x}_n-\boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_n-\boldsymbol{\mu}_2)$$

$$= -\frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{N}{2}\mathrm{Tr}\left\{\boldsymbol{\Sigma}^{-1}\mathbf{S}\right\} \qquad ($$

$$\begin{aligned}\mathbf{S} &= \frac{N_1}{N}\mathbf{S}_1 + \frac{N_2}{N}\mathbf{S}_2\\ \mathbf{S}_1 &= \frac{1}{N_1}\sum_{n\in\mathcal{C}_1}(\mathbf{x}_n-\boldsymbol{\mu}_1)(\mathbf{x}_n-\boldsymbol{\mu}_1)^{\mathrm{T}}\\ \mathbf{S}_2 &= \frac{1}{N_2}\sum_{n\in\mathcal{C}_2}(\mathbf{x}_n-\boldsymbol{\mu}_2)(\mathbf{x}_n-\boldsymbol{\mu}_2)^{\mathrm{T}}.\end{aligned}$$

- The maximum likelihood solution represents a weighted average of the covariance matrix associated with each of the two classes.

Bernoulli Distribution - Discrete Features

- We begin by considering a single binary random variable x ∈ {0, 1}.
- For example, x might describe the outcome of flipping a coin, with x = 1 representing ‘heads’, and x = 0 representing ‘tails’.
- We can imagine that this is a damaged coin so that the probability of landing heads is not necessarily the same as that of landing tails. The probability of x = 1 will be denoted by the parameter μ so that p(x = 1 | μ) = μ, where 0 ≤ μ ≤ 1, from which it follows p(x = 0 | μ)=1 − μ. The probability distribution over x can therefore be written in the form

$$Bern(x|\mu) = \mu^x(1-\mu)^{1-x}$$

- This is known as Bernoulli Distribution.
- Now suppose we have a data set D = {x_1,...,x_N } of observed values of x. We can construct the likelihood function, which is a function of μ, on the assumption that the observations are drawn independently from p(x|μ), so that

$$p(D|\mu) = \prod_{n=1}^{N} p(x_n|\mu) = \prod_{n=1}^{N}\mu^{x_n}(1-\mu)^{1-x_n}$$

- We can estimate a value for μ by maximizing the likelihood function, or equivalently by maximizing the logarithm of the likelihood. In the case of the Bernoulli distribution, the log likelihood function is given by

$$\ln p(D|\mu) = \sum_{n=1}^{N}\ln p(x_n|\mu) = \sum_{n=1}^{N}\{x_n\ln\mu + (1-x_n)\ln(1-\mu)\}$$

- Now consider the case of discrete feature values x_i. For simplicity, we begin by looking at binary feature values x_i ∈ {0, 1}. Here we will make the naive Bayes assumption in which the feature values are treated as independent, conditioned on the class C_k. Thus we have

class-conditional distributions of the form

$$p(x|C_k) = \prod_{i=1}^{D} \mu_{ki}^{x_i}(1-\mu_{ki})^{1-x_i}$$

- which contain D independent parameters for each class.
- Substituting into a_k = ln p(x|C_k)p(C_k) we get

$$a_k = \sum_{i=1}^{D}\{x_i \ln \mu_{ki} + (1-x_i)\ln(1-\mu_{ki})\} + \ln p(C_k)$$

- which again are linear functions of the input values x_i.

Example

Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

- The data tuples are described by the attributes age, income, student and credit_rating.
- The class label attribute buys_computer, has two distinct values (namely, {yes, no}).
- Let C_1 correspond to the class buys_computer = yes and C_2 correspond to buys_computer = no.
- We wish to classify X = (age = youth, income = medium, student = yes, credit_rating = fair)
- The prior probability of each class, can be computed based on the training tuples:

  P(C_1)=P(buys_computer =yes) =9/14 =0.643
  P(C_2)=P(buys_computer =no) =5/14 =0.357.
- To compute P(X|C_i), for i=1, 2, we compute the following conditional probabilities:

  P(age =youth | buys_computer =yes) = 2/9 = 0.222
  P(age =youth | buys_computer =no) =3/5 =0.600
  P(income =medium | buys_computer =yes) =4/9 =0.444
  P(income =medium | buys_computer =no) =2/5 =0.400
  P(student =yes | buys_computer =yes) =6/9 =0.667
  P(student =yes | buys_computer D=no) =1/5 =0.200
  P(credit_rating =fair | buys_computer =yes) =6/9 =0.667
  P(credit_rating =fair | buys_computer =no) =2/5 =0.400
- Using these probabilities we obtain

P(X | buys_computer =yes)= P(age =youth | buys computer = yes) *P(income = medium | buys_computer =yes)* P(student = yes | buys_computer = yes) *P(credit rating = fair | buys_computer =yes) = 0.222* 0.444 *0.667* 0.667 = 0.044

- Similarly, P(X | buys_computer = no)=0.600 *0.400* 0.200 * 0.400 = 0.019
- To find the class, C_i , that maximizes $P(X|C_i) * P(C_i)$, we compute

P(X | buys_computer = yes) *P(buys_computer = yes) = 0.044* 0.643 = 0.028

P(X | buys_computer = no) *P(buys_computer = no) =0.019* 0.357 = 0.007

- Therefore, the naive Bayesian classifier predicts buys_computer = yes for tuple X.

2.7 Probabilistic Discriminative Models

- For the two-class classification problem, we have seen that the posterior probability of class C_1 can be written as a logistic sigmoid acting on a linear function of x, for a wide choice of class-conditional distributions $p(x|C_k)$.
- Similarly, for the multiclass case, the posterior probability of class C_k is given by a softmax transformation of a linear function of x.
- For specific choices of the class-conditional densities $p(x|C_k)$, we have used maximum likelihood to determine the parameters of the densities as well as the class priors $p(C_k)$ and then used Bayes' theorem to find the posterior class probabilities.
- However, an alternative approach is to use the functional form of the generalized linear model explicitly and to determine its parameters directly by using maximum likelihood.
- The indirect approach to finding the parameters of a generalized linear model, by fitting class-conditional densities and class priors separately and then applying Bayes' theorem, represents an example of generative modelling, because we could take such a model and generate synthetic data by drawing values of x from the marginal distribution p(x).
- In the direct approach, we are maximizing a likelihood function defined through the conditional distribution $p(C_k|x)$, which represents a form of discriminative training.
- One advantage of the discriminative approach is that there will typically be fewer adaptive parameters to be determined, as we shall see shortly. It may also lead to improved predictive performance, particularly when the class-conditional density assumptions give a poor approximation to the true distributions.

Fixed Basis Functions

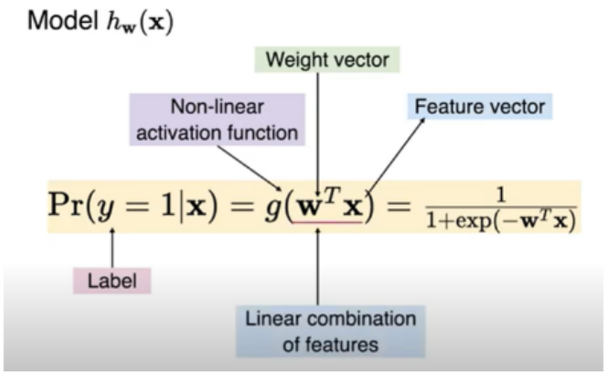
- We have considered classification models that work directly with the original input vector x.
- However, all of the algorithms are equally applicable if we first make a fixed nonlinear transformation of the inputs using a vector of basis functions φ(x).
- The resulting decision boundaries will be linear in the feature space φ, and these correspond to nonlinear decision boundaries in the original x space.
- Classes that are linearly separable in the feature space φ(x) need not be linearly separable in the original observation space x.

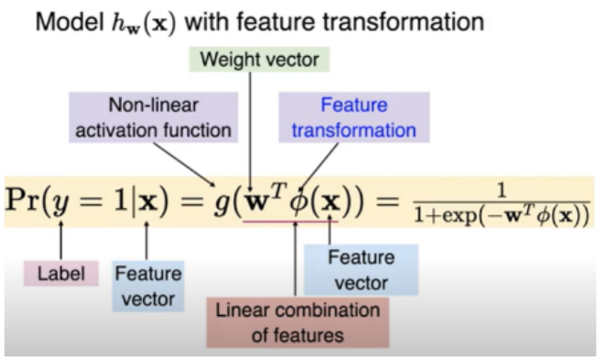
Logistic Regression

- Consider the two-class classification problem. The posterior probability of class C_1 can be written as a logistic sigmoid function:

$$p(C_1|x) = \frac{1}{1 + \exp(-w^T x)} = \sigma(w^T x)$$

- where, $p(C_2|x) = 1 - p(C_1|x)$ and we omit the bias term for clarity.
- This model is known as logistic regression.





Maximum Likelihood for Logistic Regression

- We observed a training set $\{x_n, t_n\}$, $n = 1,...,N$; $t \in \{0,1\}$
- We want to maximize the probability of getting the label right, so the likelihood function is:

$$p(t|X,w) = \prod_{n=1}^{N} [y_n^{t_n}(1-y_n)^{1-t_n}]$$

  where, $y_n = p(C_1|\varphi_n)$ and also $y_n = \sigma(w^T x_n)$
- Taking the negative log of the likelihood, we define the cross-entropy error function:

$$E(w) = -\ln p(t|X,w) = -\sum_{n=1}^{N} [t_n \ln y_n + (1-t_n)\ln(1-y_n)] = \sum_{n=1}^{N} E_n$$

- Differentiation and using the chain rule we get,

$$\frac{\mathrm{d}}{\mathrm{d}y_n} E_n = \frac{y_n - t_n}{y_n(1-y_n)}$$

$$\frac{\mathrm{d}}{\mathrm{d}w} E_n = \frac{\mathrm{dE_n}}{\mathrm{d}y_n} \frac{\mathrm{dy_n}}{\mathrm{d}w} = (y_n - t_n)x_n$$

- Therefore, we obtain

$$\nabla E(w) = \sum_{n=1}^{N} (y_n - t_n)x_n$$

- This takes exactly the same form as the gradient of the sum-of-squares error function for the linear regression model.
- Unlike linear regression, there is no closed form solution, due to non-linearity of the logistic sigmoid function.
- The error function is convex and can be optimized using standard gradient-based or more advanced optimization techniques.

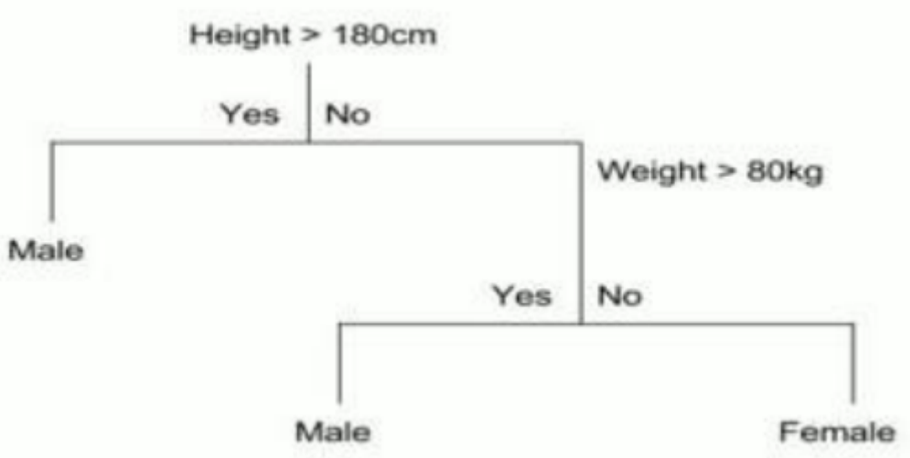
Chapter-3 Decision Trees & Ensembles

3.1 Decision Tree Learning

- Construction of decision tree from class-labeled training tuples
- It is a flow-chart like structure, where each internal node (non-leaf) denotes a test on an attribute, each branch represents the outcome of the test, and each leaf node (terminal) holds a class label.
- The population or sample is split into two or more homogeneous sets (sub-populations) based on most significant splitter/differentiator in input variables.

Example

- Given a dataset of two inputs (x) of height in centimeters and weight in kilograms, the output is the sex as male or female.
- The tree can be stored into a file as a graph or a set of rules.



- For example, below is the above decision tree as a set of rules.

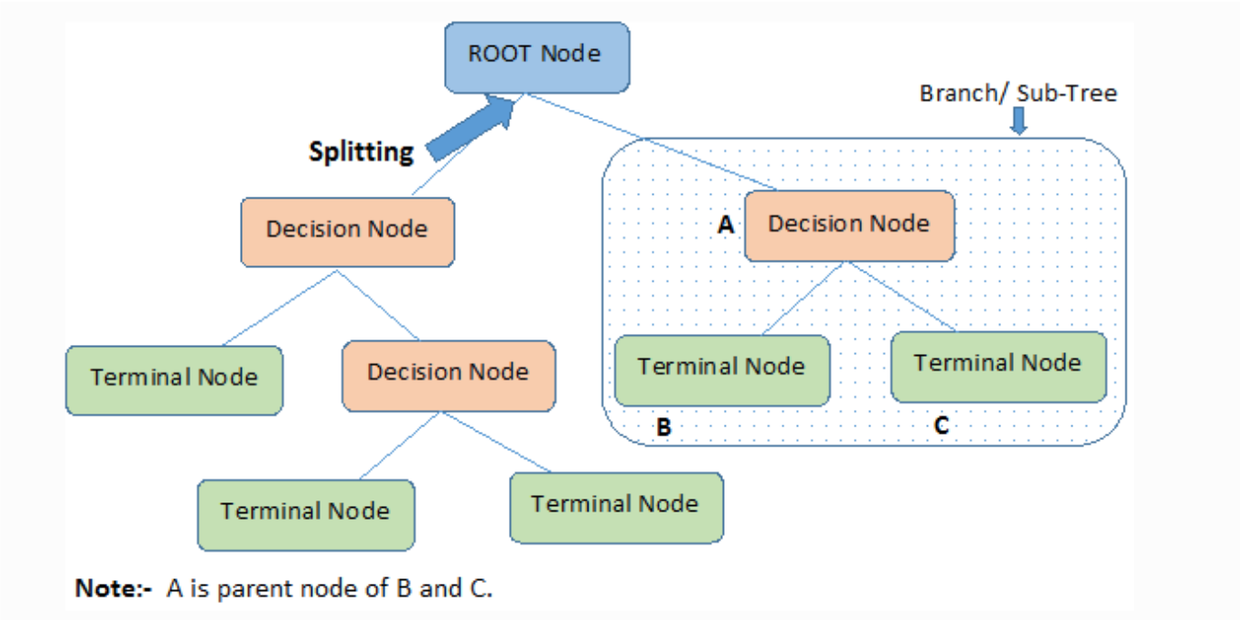
```
- If Height > 180 cm Then Male
- If Height <= 180 cm AND Weight > 80 kg Then Male
- If Height <= 180 cm AND Weight <= 80 kg Then Female
```

- For example, given the input of [height = 160 cm, weight = 65 kg], we would traverse the above tree as follows:

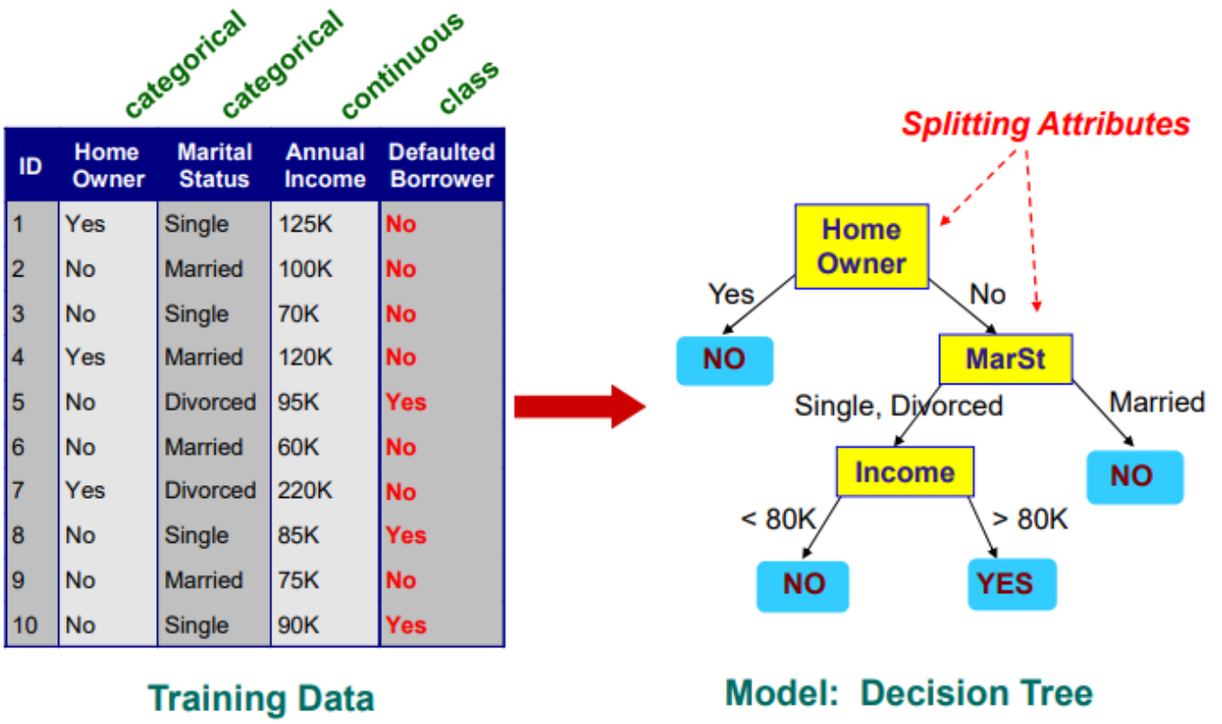
```
Height > 180 cm: No
Weight > 80 kg: No
Therefore: Female
```

Important Terminology

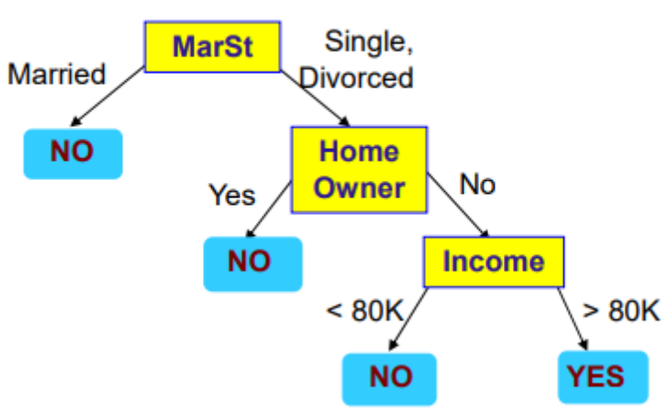
1. **Root Node:** It is the topmost node in the tree. It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called the decision node.
4. **Leaf / Terminal Node:** Nodes do not split is called Leaf or Terminal node.



Example of a decision tree

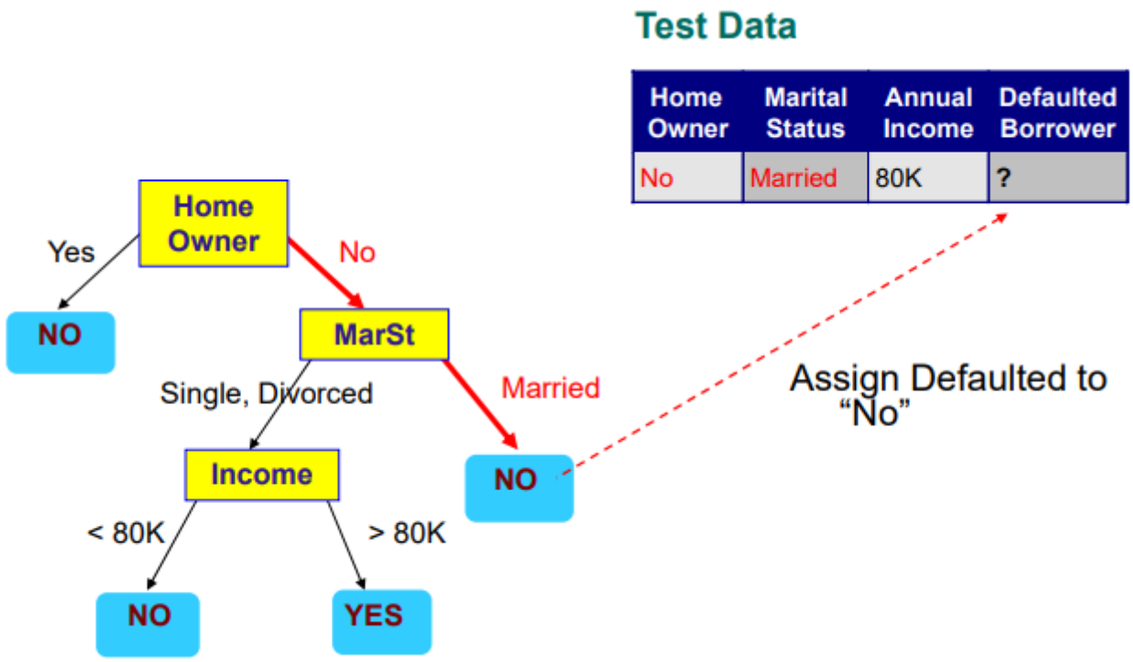


- The classification problem can be solved by asking a series of carefully crafted questions about the attributes of the test record.
- The series of questions and their possible answers can be organized in the form of a decision tree.
- There could be more than one tree that fits the same data.

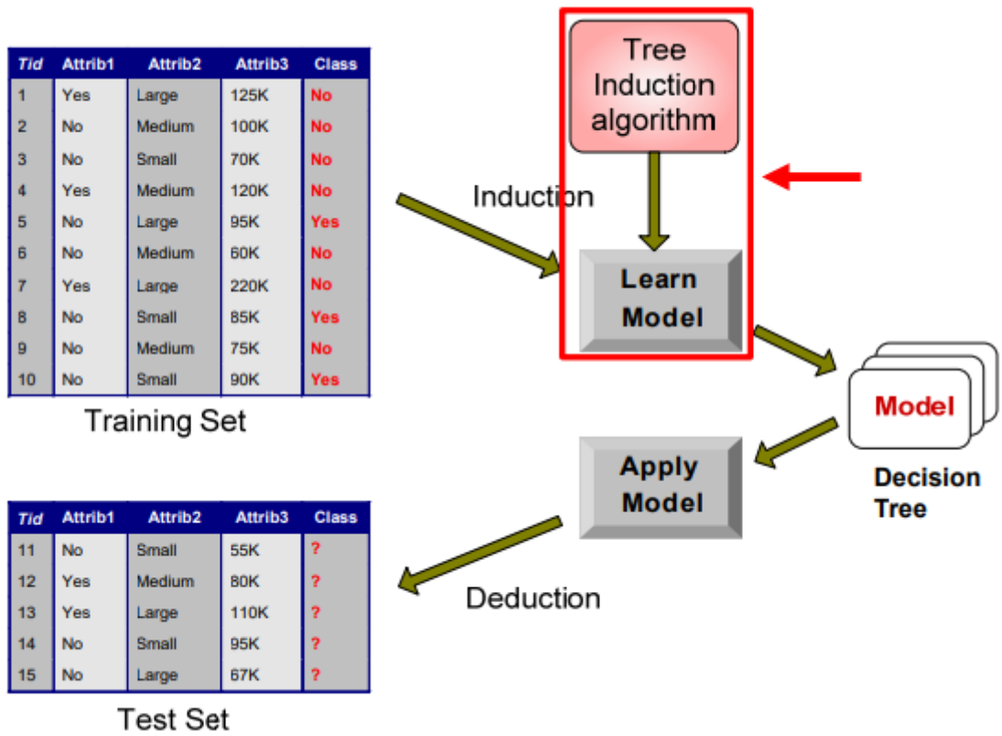


Applying Model to Test Data

- Starting from the root node of the tree, we apply the test condition to the record and follow the appropriate branch based on the outcome of the test.
- This will lead us either to another internal node, for which a new test is applied, or to a leaf node.
- The class label associated with the leaf node is then assigned to the record.



Decision Tree Classification Task



3.2 Decision Tree Induction

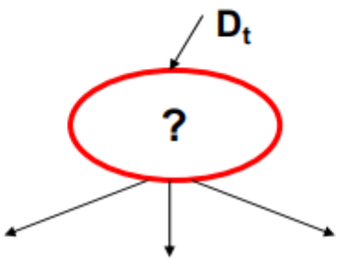
Many Algorithms:

- Hunt's Algorithm (one of the earliest)
- CART
- ID3, C4.5
- SLIQ,SPRINT

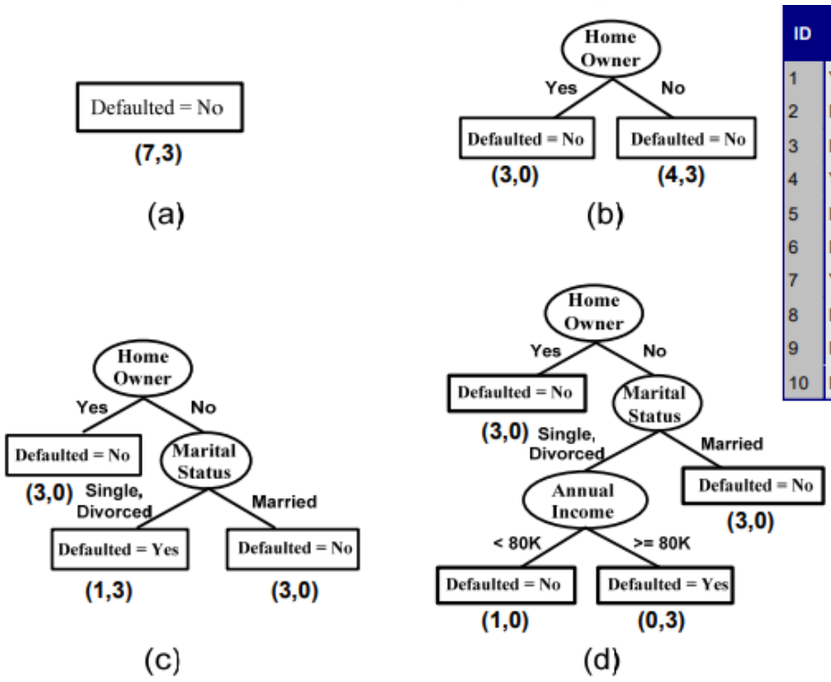
Hunt's Algorithm

- In Hunt's Algorithm, a decision tree is grown in a recursive fashion by partitioning the training records into successively purer subsets.
- Let D_t be the set of training records that reach a node t and y = $\{y_1,...y_c\}$ be the class labels.
- If D_t contains records that belong the same class y_t , then t is a leaf node labeled as y_t
- If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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



- Consider the problem of predicting whether a loan applicant will repay their loan obligations or become delinquent, subsequently defaulting on their loan.



ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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

Design Issues of Decision Tree Induction

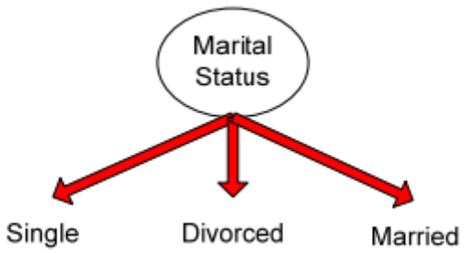
- Greedy Strategy: Split records based on an attribute test that optimizes certain criterion.
- Issues:
  - Determine how to split the records
    - How to specify the attribute test condition
    - How to determine the best split
  - Determine when to stop splitting

How to Specify Attribute Test Condition

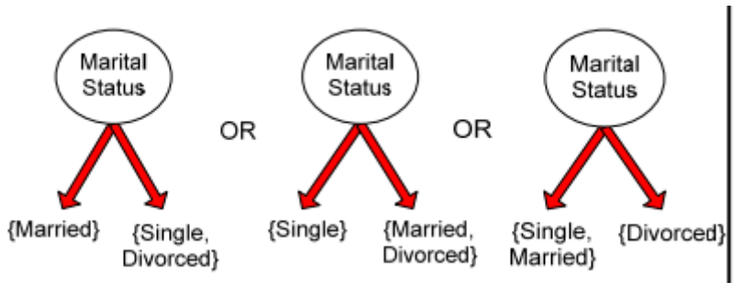
- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting Based on Nominal Attributes

1. Multi-way Split: Use as many partitions as distinct values

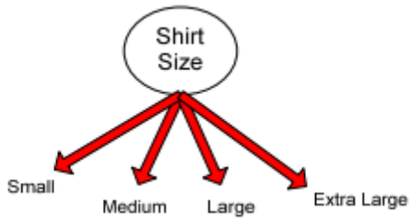


2. Binary Split: Divides values into two subsets and need to find optimal partitioning

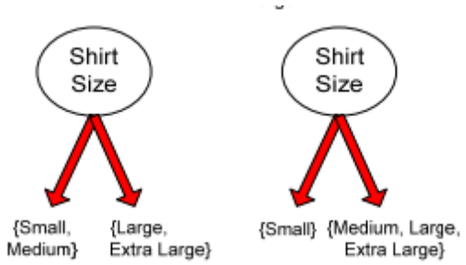


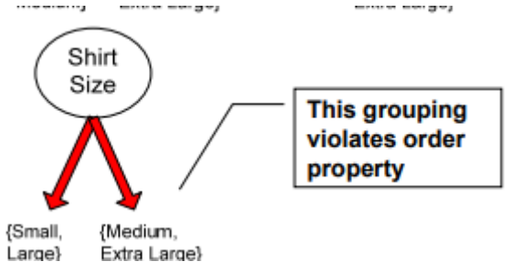
Splitting Based on Ordinal Attributes

1. Multi-way Split: Use as many partitions as distinct values



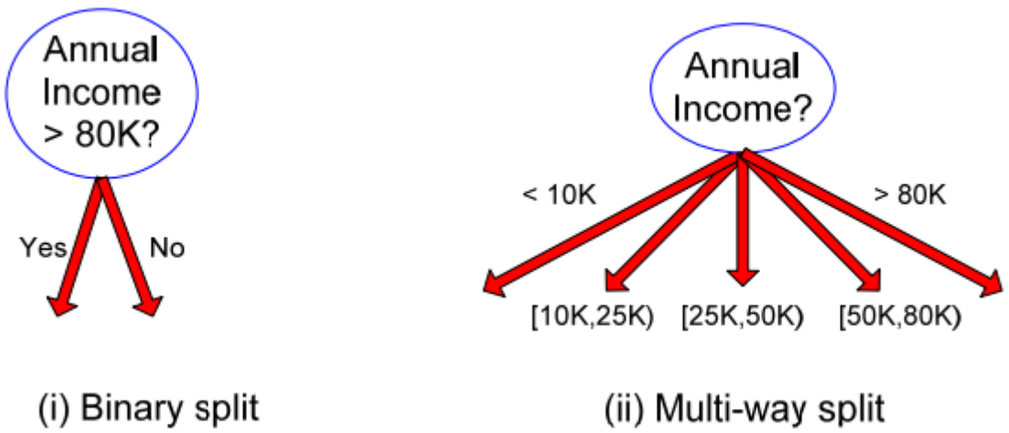
2. Binary Split: Divides values into two subsets and need to find optimal partitioning





Splitting Based on Continuous Attributes

- Different ways of handling:
  - Binary decision: (A<v) or (A≥v)
    - Consider all possible splits and find the best cut
    - Can be more compute intensive
  - For multiway split:
    - Consider all possible ranges of continuous values and apply the discretization strategies
    - After discretization, a new ordinal value will be assigned to each discretized interval

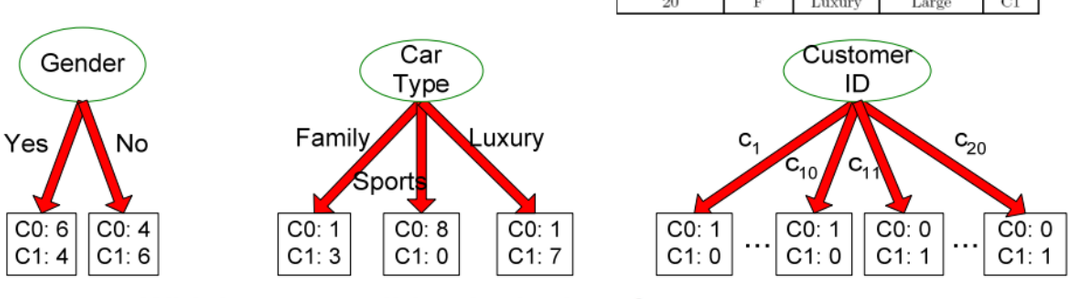


Determining Best Split

- Greedy Approach: Nodes with homogeneous class distribution are preferred.
- Need a measure of node impurity

Before Splitting: 10 records of class 0, 10 records of class 1

Customer Id	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1



Which test condition is the best?

Measures of Node Impurity

1. Gini Index

$$Gini\ Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

where $p_i(t)$ is the frequency of class i at node t, and c is the total number of classes

2. Entropy

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t)$$

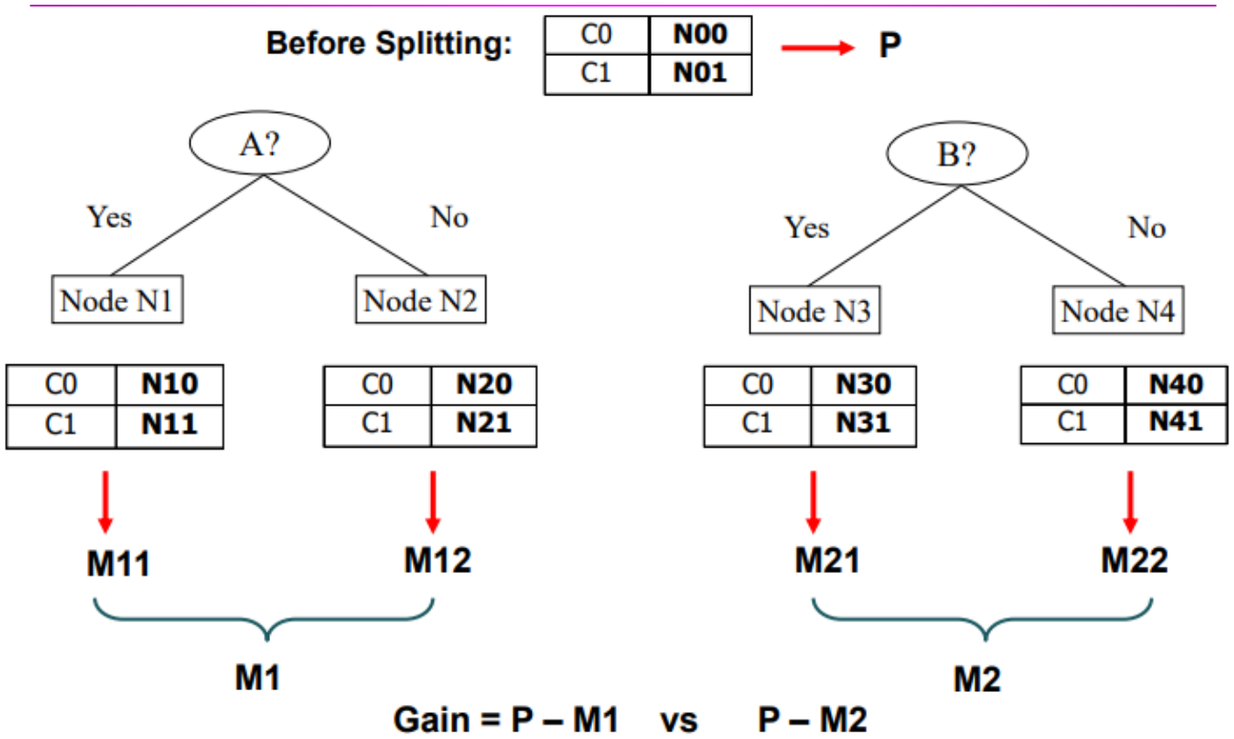
3. Misclassification Error

$$Classification\ Error = 1 - max[p_i(t)]$$

- Compute impurity measure (P) before splitting
- Compute impurity measure (M) after splitting
  - Compute impurity measure of each child node
  - M is the weighted impurity of child nodes
- Choose the attribute test condition that produces the highest gain

  $$Gain = P - M$$

  or equivalently, lowest impurity measure after splitting (M)



3.3 Measures of Impurity

1. GINI Index

For a node t,

$$Gini\ index = 1 - \sum_{i=0}^{(c-1)} p_i(t)^2$$

For a 2 class problem (p, 1-p):

$$GINI = 1 - p^2 - (1-p)^2 = 2p(1-p)$$

C1	0
C2	6
Gini=0.000	

C1	1
C2	5
Gini=0.278	

C1	2
C2	4
Gini=0.444	

C1	3
C2	3
Gini=0.500	

Computing Gini Index on a single node

C1	0
C2	6

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Gini = 1 – P(C1)² – P(C2)² = 1 – 0 – 1 = 0

C1	1
C2	5

P(C1) = 1/6 P(C2) = 5/6

Gini = 1 – (1/6)² – (5/6)² = 0.278

C1	2
C2	4

P(C1) = 2/6 P(C2) = 4/6

Gini = 1 – (2/6)² – (4/6)² = 0.444

Computing Gini Index for a collection of nodes

When a node p is split into k partitions (children)

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

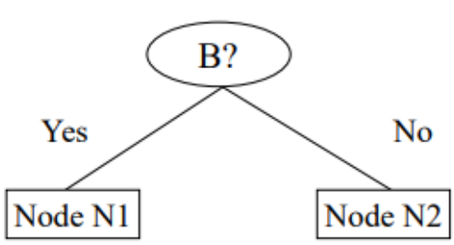
where,

n_i = number of records at child i

n = number of records at parent node p

Binary Attributes - Computing Gini Index

- Splits into two partitions (child nodes)
- Effect of Weighing partitions – Larger and purer partitions are sought



	Parent
C1	7
C2	5
Gini = 0.486	

Gini(N1)
= 1 – (5/6)² – (1/6)²
= 0.278

Gini(N2)
= 1 – (2/6)² – (4/6)²
= 0.444

	N1	N2
C1	5	2
C2	1	4
Gini=0.361		

Weighted Gini of N1 N2
= 6/12 * 0.278 +
6/12 * 0.444
= 0.361

Gain = 0.486 – 0.361 = 0.125

Categorical Attributes - Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	0.163		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	0.468	

	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

2. Entropy

Entropy at a given node t

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t)$$

where $p_i(t)$ is the frequency of class i at node t, and c is the total number of classes

Computing Entropy of a single node

C1	0
C2	6

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	1
C2	5

P(C1) = 1/6 P(C2) = 5/6

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (1/6) = 0.65

C1	2
C2	4

P(C1) = 2/6 P(C2) = 4/6

Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92

Computing Information Gain after splitting

- Information Split:

$$Gain_{split} = Entropy(p) - \sum_{i=1}^{k} \frac{n_i}{n} Entropy(i)$$

- Parent Node, p is split into k partitions (children)
- n_i is number of records in child node i
- Choose the split that achieves most reduction (maximizes GAIN)
- Used in the ID3 and C4.5 decision tree algorithms

3. Classification Error

Classification Error at a node t,

$$Classification\ Error = 1 - \max_i [p_i(t)]$$

Computing Error of a single node

C1	0
C2	6

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	1
C2	5

P(C1) = 1/6 P(C2) = 5/6

Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

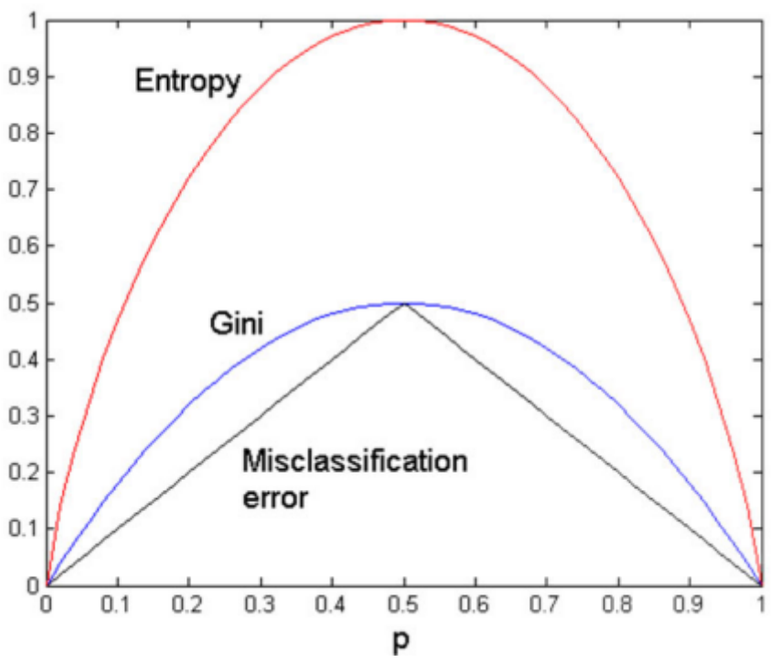
C1	2
C2	4

P(C1) = 2/6 P(C2) = 4/6

Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Comparison among Impurity Measures

- For 2-class problem



- All three measures attain their maximum value when the class distribution is uniform (p = 0.5)
- The minimum values for the measures are attained when all the records belong to the same class

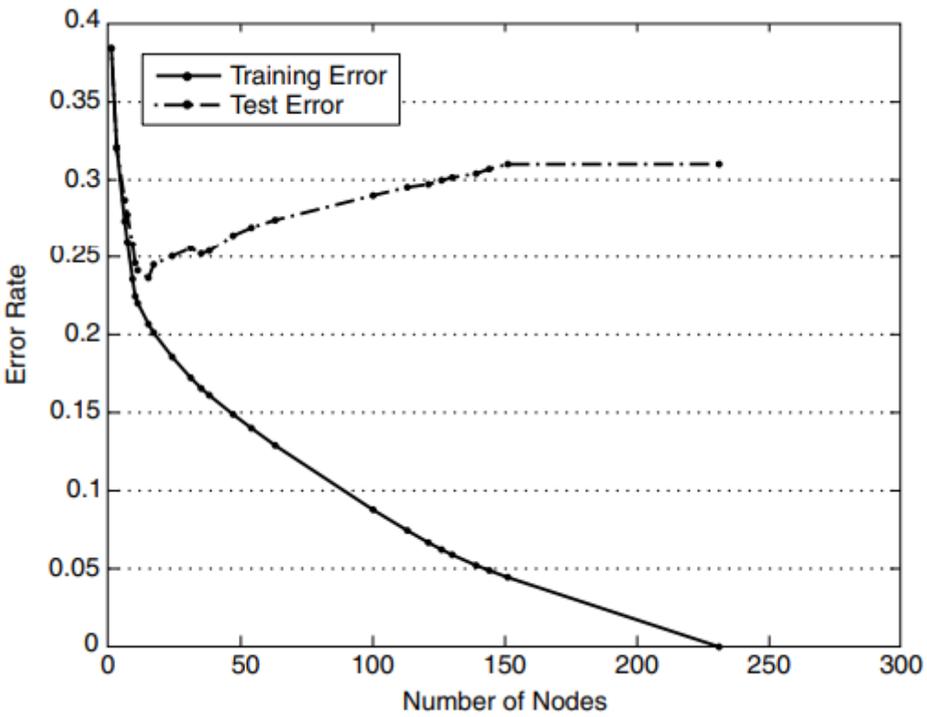
Decision Tree Based Classification

- Advantages
  - Relatively inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Robust to noise (especially when methods to avoid overfitting are employed)
  - Can easily handle redundant attributes
  - Can easily handle irrelevant attributes (unless the attributes are interacting)
- Disadvantages
  - Due to the greedy nature of splitting criterion, interacting attributes (that can distinguish between classes together but not individually) may be passed over in favor of other attributed that are less discriminating.

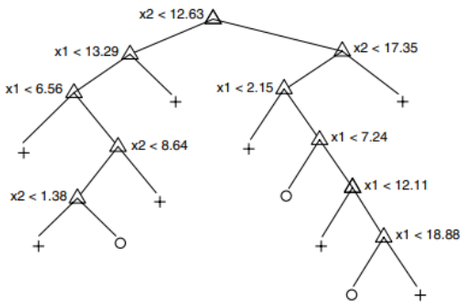
- Each decision boundary involves only a single attribute

3.4 Model Overfitting and Underfitting

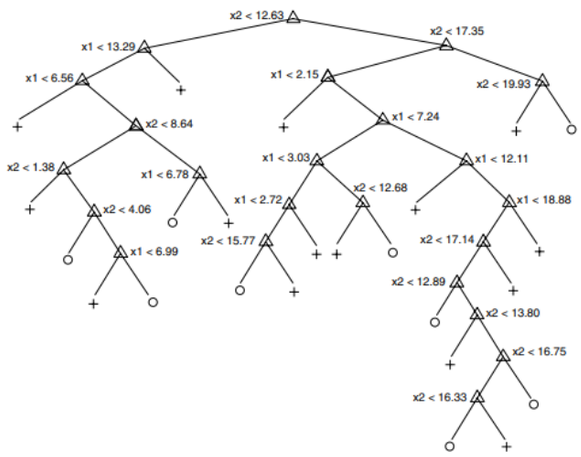
- Errors committed by a classification model are generally divided into
  - Training Errors: misclassification on training set records
  - Generalization Errors (Testing Errors): errors made on testing set
- Good model has low training error and low generalization error
- Overfitting: model has low training error rate,, but high generalization errors
- The training and test error rates of the model are large when the size of the tree is very small. This situation is known as model overfitting.
  - Underfitting occurs because the model has yet to learn the true structure of the data. As a result, it performs poorly on both the training and test sets.
- As the number of nodes increases, the tree will have fewer training and test errors. However, once the tree becomes too large, its test error begins to increase even though its training error rate continues to decrease. This is known as model overfitting.



- The training error of a model can be reduced by increasing the model complexity.
- For example, the leaf nodes of the tree can be expanded until it perfectly fits the training data.
  - Although the training error for such a complex tree is zero, the test error can be large because the tree may contain nodes that accidently fit some of the noise points in the training data.
  - Such nodes can degrade the performance of the tree because they do not generalize well to the test examples.



(a) Decision tree with 11 leaf nodes.



(b) Decision tree with 24 leaf nodes.

- The tree that contains the smaller number of nodes has a higher training error rate, but a lower test error rate compared to the more complex tree.

Reasons for Overfitting

1. Presence of noise
2. Lack of representative samples

Overfitting due to presence of noise

Table 4.3. An example training set for classifying mammals. Class labels with asterisk symbols represent mislabeled records.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
porcupine	warm-blooded	yes	yes	yes	yes
cat	warm-blooded	yes	yes	no	yes
bat	warm-blooded	yes	no	yes	no*
whale	warm-blooded	yes	no	no	no*
salamander	cold-blooded	no	yes	yes	no
komodo dragon	cold-blooded	no	yes	no	no
python	cold-blooded	no	no	yes	no
salmon	cold-blooded	no	no	no	no
eagle	warm-blooded	no	no	no	no
guppy	cold-blooded	yes	no	no	no

Table 4.4. An example test set for classifying mammals.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
human	warm-blooded	yes	no	no	yes
pigeon	warm-blooded	no	no	no	no
elephant	warm-blooded	yes	yes	no	yes
leopard shark	cold-blooded	yes	no	no	no
turtle	cold-blooded	no	yes	no	no
penguin	cold-blooded	no	no	no	no
eel	cold-blooded	no	no	no	no
dolphin	warm-blooded	yes	no	no	yes
spiny anteater	warm-blooded	no	yes	yes	yes
gila monster	cold-blooded	no	yes	yes	no

- Tree perfectly fits the training data.
- Test error rate is 30% -> Both humans and dolphins were misclassified as nonmammals because their attribute values for Body Temperature, Gives Birth, and Four-legged are identical to the mislabeled records in the training set.

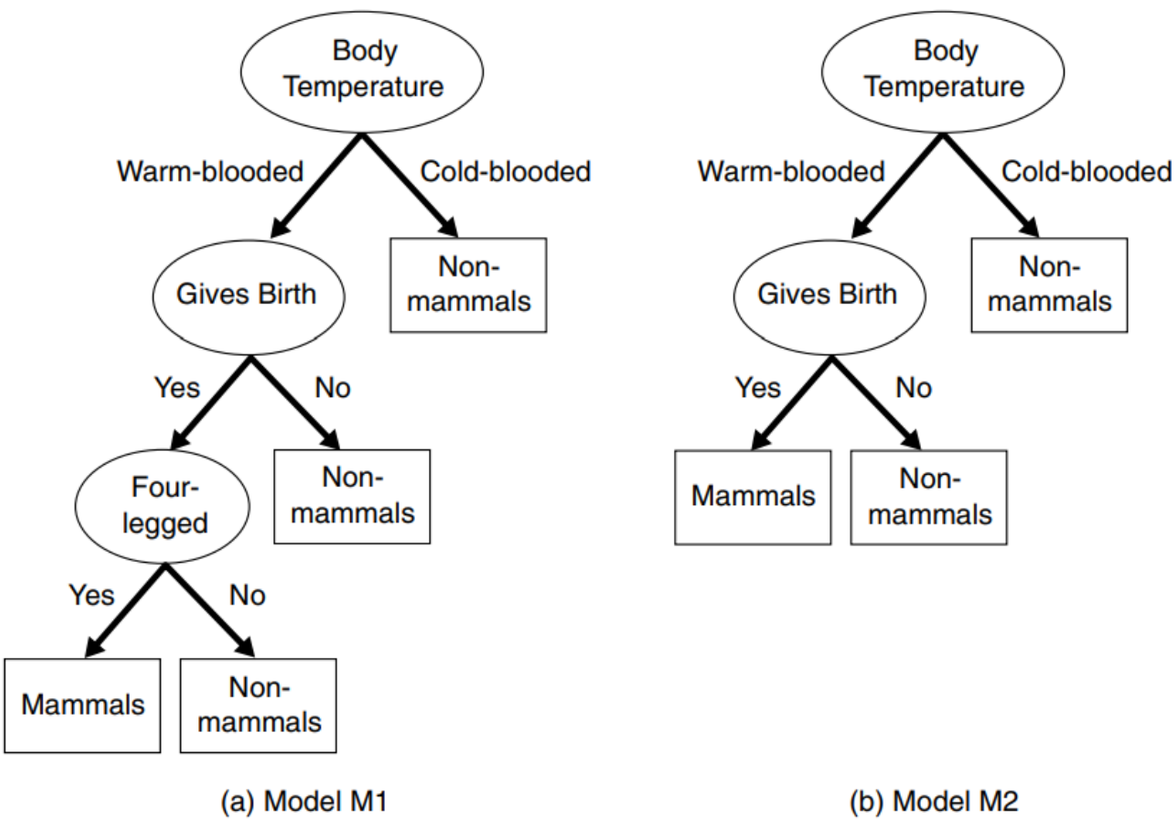


Figure 4.25. Decision tree induced from the data set shown in Table 4.3.

- M1 has overfitted the training data because there is a simpler model with lower error rate on the test set.
- The Four-legged attribute test condition in model M1 is spurious because it fits the mislabeled training records, which leads to the misclassification of records in the test set

Overfitting due to lack of representative samples

- Models that make their classification decisions based on a small number of training records are also susceptible to overfitting.

Table 4.5. An example training set for classifying mammals.

Name	Body Temperature	Gives Birth	Four-legged	Hibernates	Class Label
salamander	cold-blooded	no	yes	yes	no
guppy	cold-blooded	yes	no	no	no
eagle	warm-blooded	no	no	no	no
poorwill	warm-blooded	no	no	yes	no
platypus	warm-blooded	no	yes	yes	yes

- Humans, elephants, and dolphins are misclassified because the decision tree classifies all warm-blooded vertebrates that do not hibernate as non-mammals.
- The tree arrives at this classification decision because there is only one training record, which is an eagle, with such characteristics.

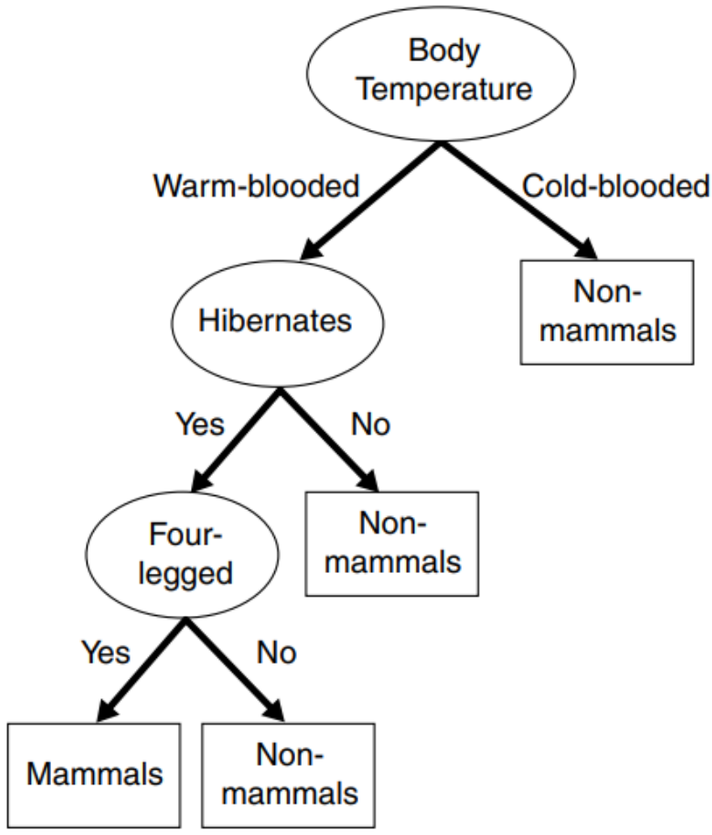


Figure 4.26. Decision tree induced from the data set shown in Table 4.5.

Handling Overfitting in Decision Trees

- Two strategies for avoiding model overfitting in the context of decision tree induction.
  - Pre-pruning (Early Stopping)
  - Post-pruning

Pre-pruning

- The tree-growing algorithm is halted before generating a fully grown tree that perfectly fits the entire training data.
- To do this, a more restrictive stopping condition must be used; e.g., stop expanding a leaf node when the observed gain in impurity measure (or improvement in the estimated generalization error) falls below a certain threshold.
- The advantage of this approach is that it avoids generating overly complex subtrees that overfit the training data.
- Nevertheless, it is difficult to choose the right threshold for early termination.

Post-pruning

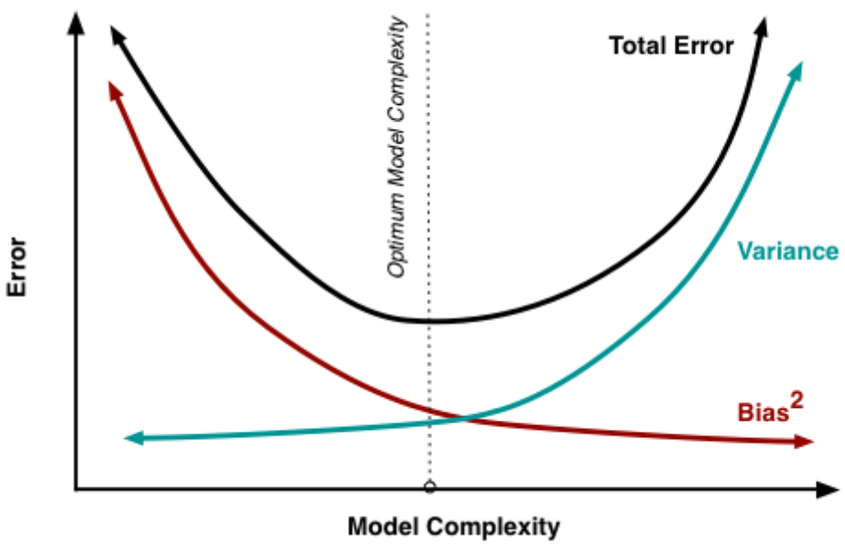
- In this approach, the decision tree is initially grown to its maximum size.
- This is followed by a tree-pruning step, which proceeds to trim the fully grown tree in a bottom-up fashion.
- Trimming can be done by replacing a subtree with
  - a new leaf node whose class label is determined from the majority class of records affiliated with the subtree, or
  - the most frequently used branch of the subtree.
- The tree-pruning step terminates when no further improvement is observed.
- Post-pruning tends to give better results than pre-pruning because it makes pruning decisions based on a fully grown tree, unlike pre-pruning, which can suffer from premature termination of the tree-growing process.

- However, for post-pruning, the additional computations needed to grow the full tree may be wasted when the subtree is pruned.

3.5 Bias-Variance Tradeoff

- The goal of any supervised machine learning algorithm is to best estimate the mapping function (f) for the output variable (Y) given the input data (X).
- The mapping function is often called the target function because it is the function that a given supervised machine learning algorithm aims to approximate.
- The prediction error for any machine learning algorithm can be broken down into three parts:
  - Bias Error
  - Variance Error
  - Irreducible Error
- The irreducible error cannot be reduced regardless of what algorithm is used.
  - It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

Bias and Variance as a Function of Model Complexity



Bias Error

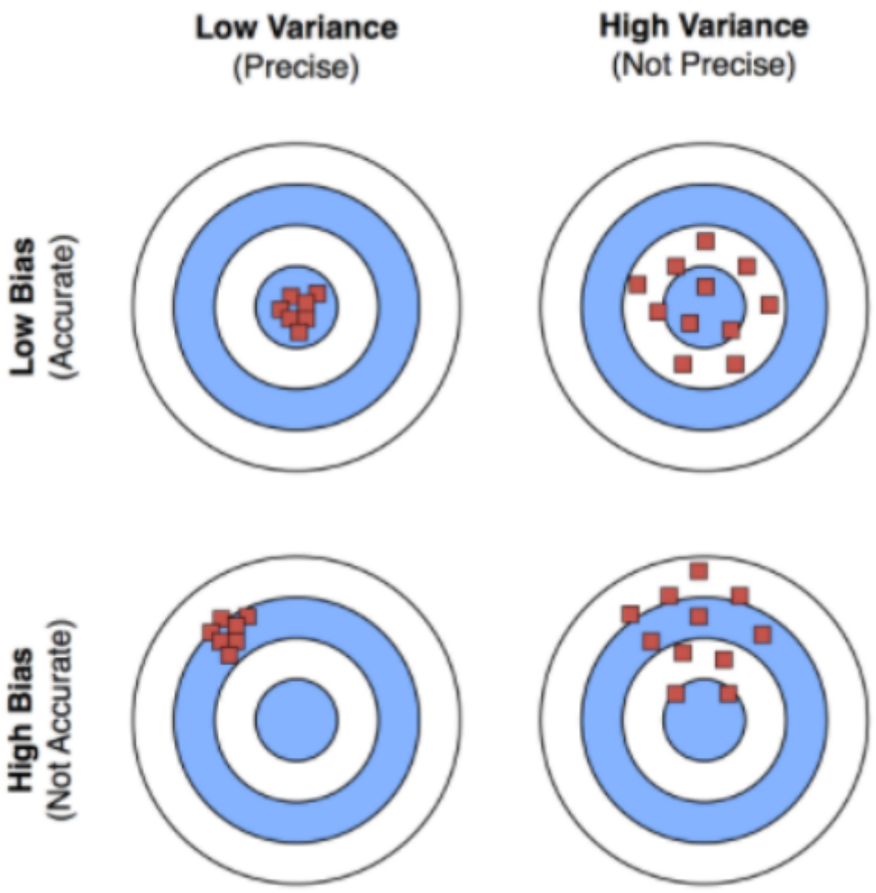
- Bias are the simplifying assumptions made by a model to make the target function easier to learn.
- Generally, linear algorithms have a high bias making them fast to learn and easier to understand but generally less flexible.
- In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.
- **Low Bias**: Suggests less assumptions about the form of the target function.
- **High-Bias**: Suggests more assumptions about the form of the target function.
- Examples of low-bias machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.
- Examples of high-bias machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.

Variance Error

- Variance is the amount that the estimate of the target function will change if different training data was used.
- The target function is estimated from the training data by a machine learning algorithm, so we should expect the algorithm to have some variance.
- Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables.
- **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset.
- **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset.
- Generally, nonlinear machine learning algorithms that have a lot of flexibility have a high variance. For example, decision trees have a high variance, that is even higher if the trees are not pruned before use.
- Examples of low-variance machine learning algorithms include: Linear Regression, Linear Discriminant Analysis and Logistic Regression.
- Examples of high-variance machine learning algorithms include: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

Graphical Definition

- We can create a graphical visualization of bias and variance using a bulls-eye diagram.
- Imagine that the center of the target is a model that perfectly predicts the correct values. As we move away from the bulls-eye, our predictions get worse and worse. Imagine we can repeat our entire model building process to get a number of separate hits on the target.
    - Each hit represents an individual realization of our model, given the chance variability in the training data we gather.
    - Sometimes we will get a good distribution of training data so we predict very well and we are close to the bulls-eye, while sometimes our training data might be full of outliers or non-standard values resulting in poorer predictions.
    - These different realizations result in a scatter of hits on the target.
- Underfitting happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data.
- Overfitting happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance.



- The bias-variance tradeoff is a central problem in supervised learning. Ideally, one wants to choose a model that both accurately captures the regularities in its training data, but also generalizes well to unseen data.
- Unfortunately, it is typically impossible to do both simultaneously.
  - High-variance learning methods may be able to represent their training set well, but are at risk of overfitting to noisy or unrepresentative training data.
  - In contrast, algorithms with high bias typically produce simpler models that may fail to capture important regularities (i.e. underfit) in the data.
- The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.
- Usually there is a trend as follows:
  - Linear machine learning algorithms often have a high bias but a low variance.
  - Nonlinear machine learning algorithms often have a low bias but a high variance.
- The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

3.6 Ensembles

- Ensemble learning refers to algorithms that combine the predictions from two or more models.
- In broad terms, using ensemble methods is about combining models such that the ensemble has a better performance than an individual model on average.
- The three main classes of ensemble learning methods are bagging, stacking, and boosting.

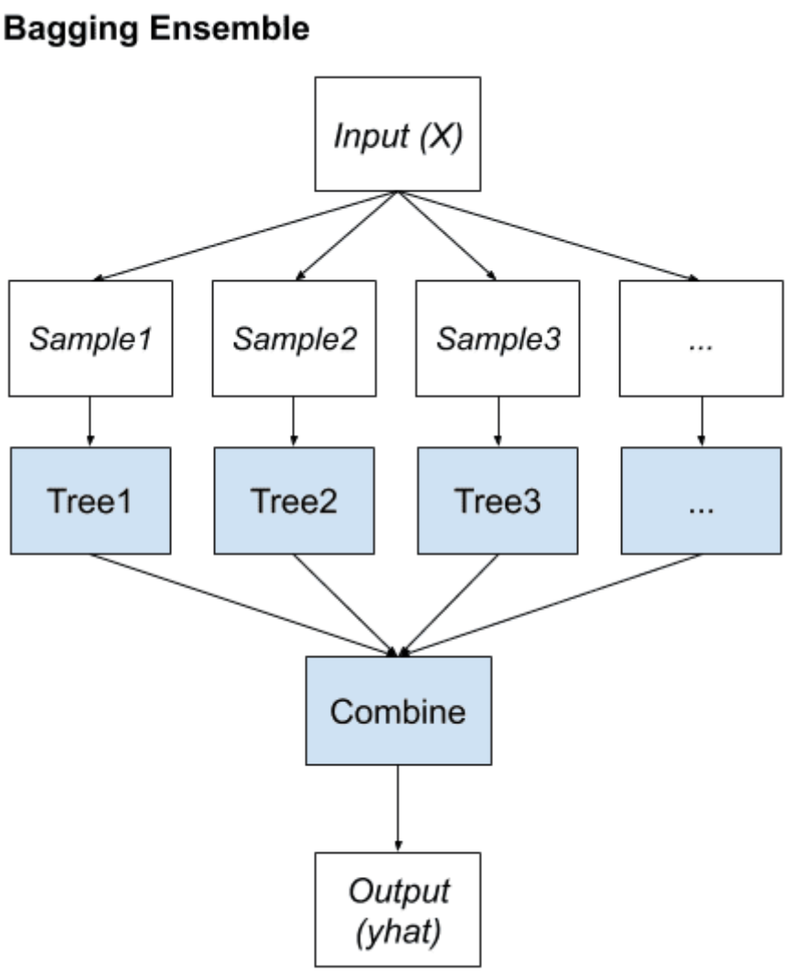
Different Ensemble Models

- Bagging involves fitting many decision trees on different samples of the same dataset and averaging the predictions.
- Stacking involves fitting many different models types on the same data and using another model to learn how to best combine the predictions.

- Boosting involves adding ensemble members sequentially that correct the predictions made by prior models and outputs a weighted average of the predictions.

Bagging

- The name Bagging came from the abbreviation of Bootstrap AGGregatING.
- This typically involves using a single machine learning algorithm, almost always an unpruned decision tree, and training each model on a different sample of the same training dataset.
- The predictions made by the ensemble members are then combined using simple statistics, such as voting or averaging.



Bootstrap sampling

- Bagging adopts the bootstrap distribution for generating different base learners. In other words, it applies bootstrap sampling to obtain the data subsets for training the base learners.
- A bootstrap sample is a sample of size n drawn with replacement from an original training set D with |D| = n.
- Consequently, some training examples are duplicated in each bootstrap sample, and some other training examples do not appear in a given bootstrap sample at all. Usually, we refer to these examples as "out-of-bag sample"

Algorithm 1 Bagging

1: Let n be the number of bootstrap samples
2:
3: **for** i=1 to n **do**
4: Draw bootstrap sample of size m, $\mathcal{D}_i$
5: Train base classifier h_i on $\mathcal{D}_i$
6: $\hat{y} = mode\{h_1(\mathbf{x}), ..., h_n(\mathbf{x})\}$

- In the limit, there are approximately 63.2% unique examples in a given bootstrap sample.
- Consequently, 37.2% examples from the original training set will not appear in a given bootstrap sample at all.
- If we sample from a uniform distribution, we can compute the probability that a given example from a dataset of size n is not drawn as a bootstrap sample as

$$P(\text{not chosen}) = \left(1 - \frac{1}{n}\right)^n,$$

which is asymptotically equivalent to

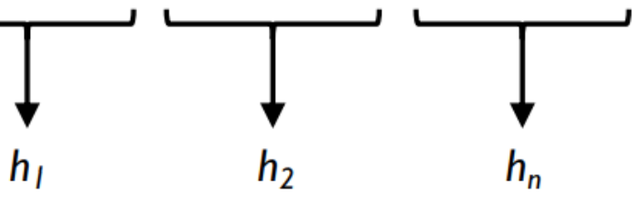
$$\frac{1}{e} \approx 0.368 \quad \text{as} \quad n \to \infty.$$

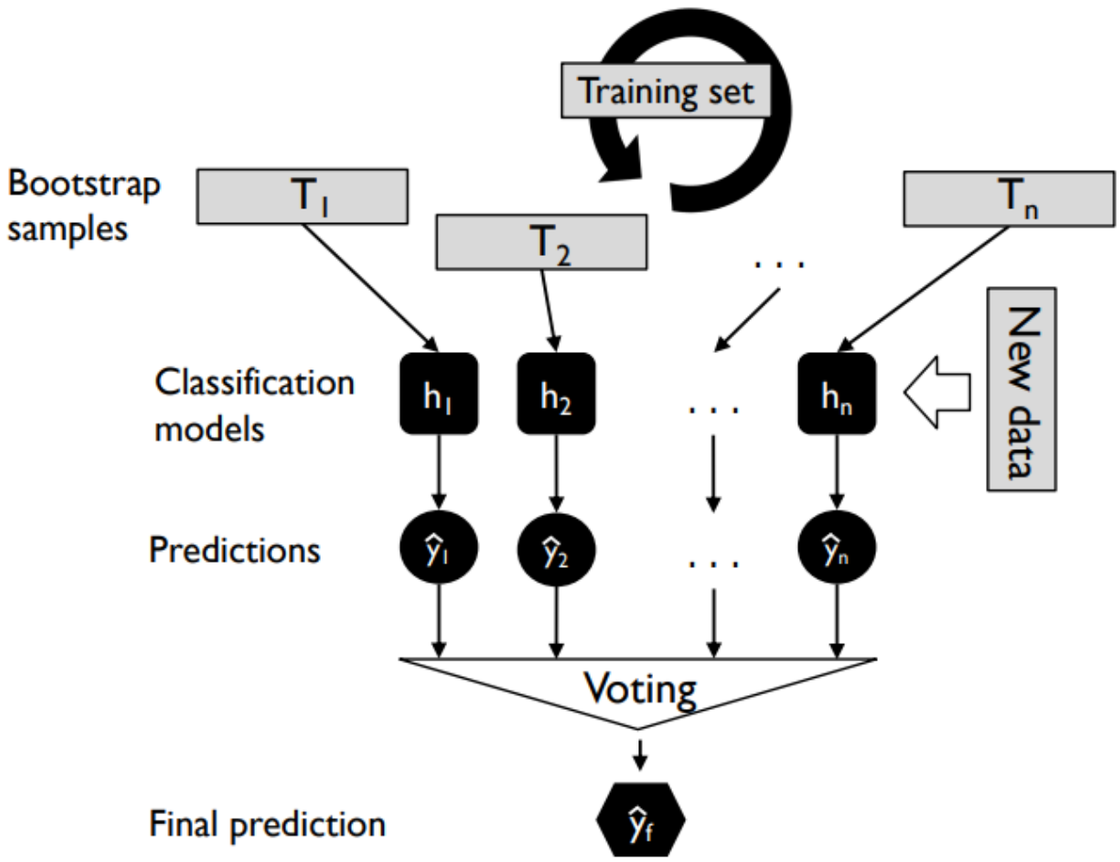
Vice versa, we can then compute the probability that a sample is chosen as

$$P(\text{chosen}) = 1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632.$$

Bootstrapping in the context of Bagging

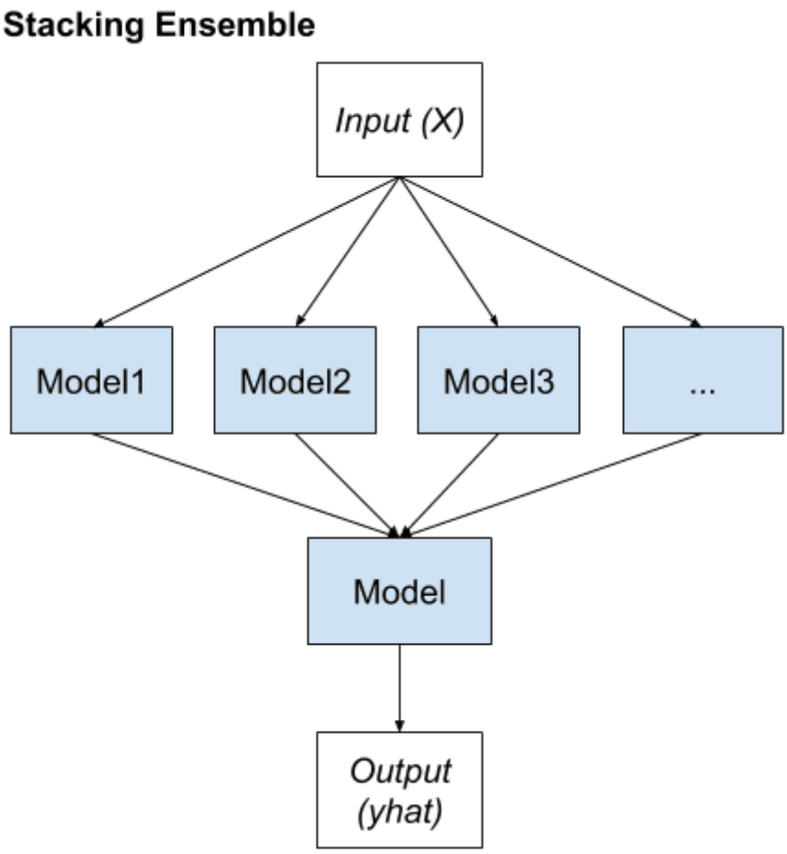
Training example indices	Bagging round 1	Bagging round 2	…
1	2	7	…
2	2	3	…
3	1	2	…
4	3	1	…
5	7	1	…
6	2	7	…
7	4	7	…





Stacking

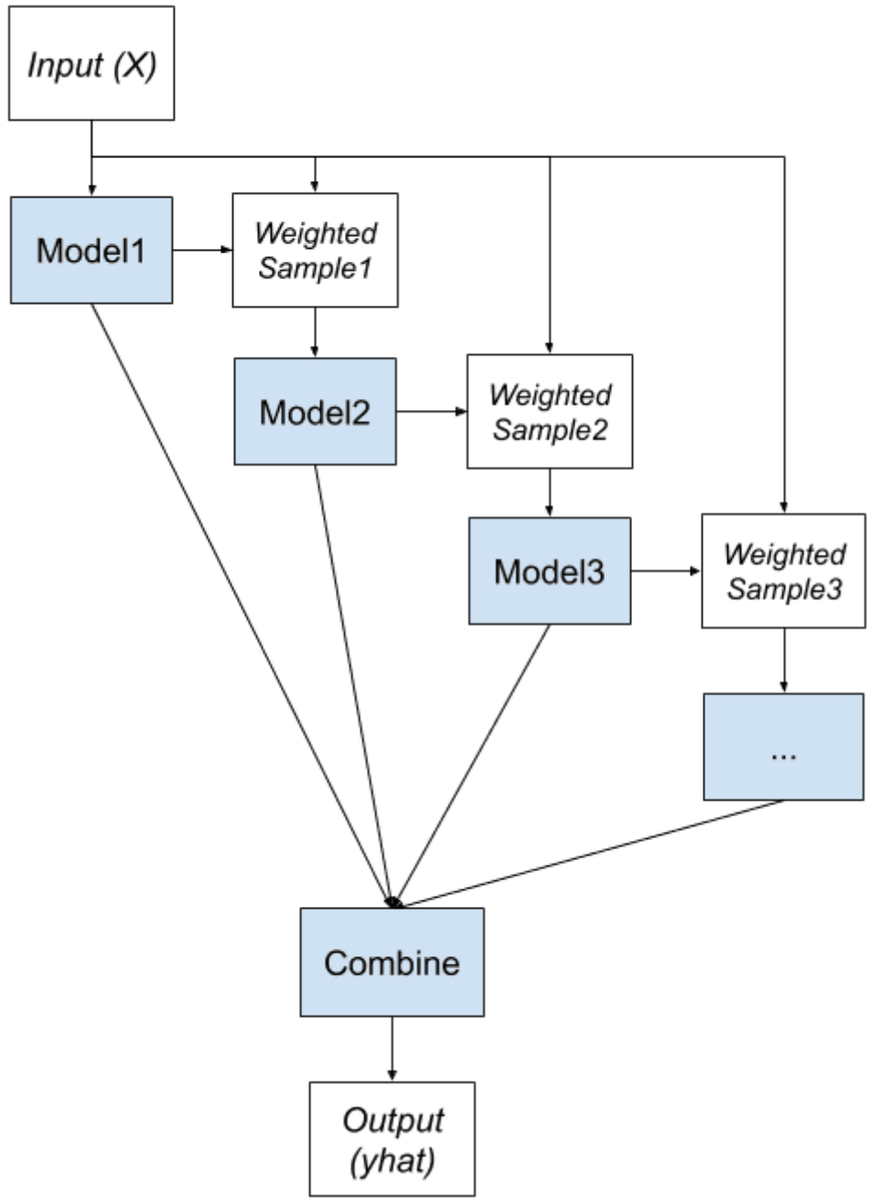
- Stacking is a general procedure where a learner is trained to combine the individual learners. Here, the individual learners are called the first-level learners, while the combiner is called the second-level learner, or meta-learner.



Boosting

- In boosting, the training dataset for each subsequent classifier increasingly focuses on instances misclassified by previously generated classifiers.
- The key property of boosting ensembles is the idea of correcting prediction errors.
- The models are fit and added to the ensemble sequentially such that the second model attempts to correct the predictions of the first model, the third corrects the second model, and so on. lead
- This typically involves the use of very simple decision trees that only make a single or a few decisions, referred to in boosting as weak learners.
- The predictions of the weak learners are combined using simple voting or averaging, although the contributions are weighed proportional to their performance or capability. The objective is to develop a so-called "strong-learner" from many purpose-built "weak-learners."

Boosting Ensemble



AdaBoost, or Adaptive Boost

- The way boosting generally works is you build multiple models on subsets of the dataset in a sequential manner, where each model learns from the previous models' mistakes. After you build all the weak models, you then combine their predictions to get a "strong model".
- AdaBoost, short for Adaptive Boosting is one of the early successful algorithms within the Boosting branch of machine learning, and is used specifically for binary classification.

Adaboost Algorithm

1. For a dataset with N number of samples, we initialize the weight of each data point with $w_i = \frac{1}{N}$.

2. For $m = 1$ to M:

(a) Sample the dataset using the weights $w_i^{(m)}$ to obtain training samples x_i.

(b) Fit a classifier K_m using all the training samples x_i.

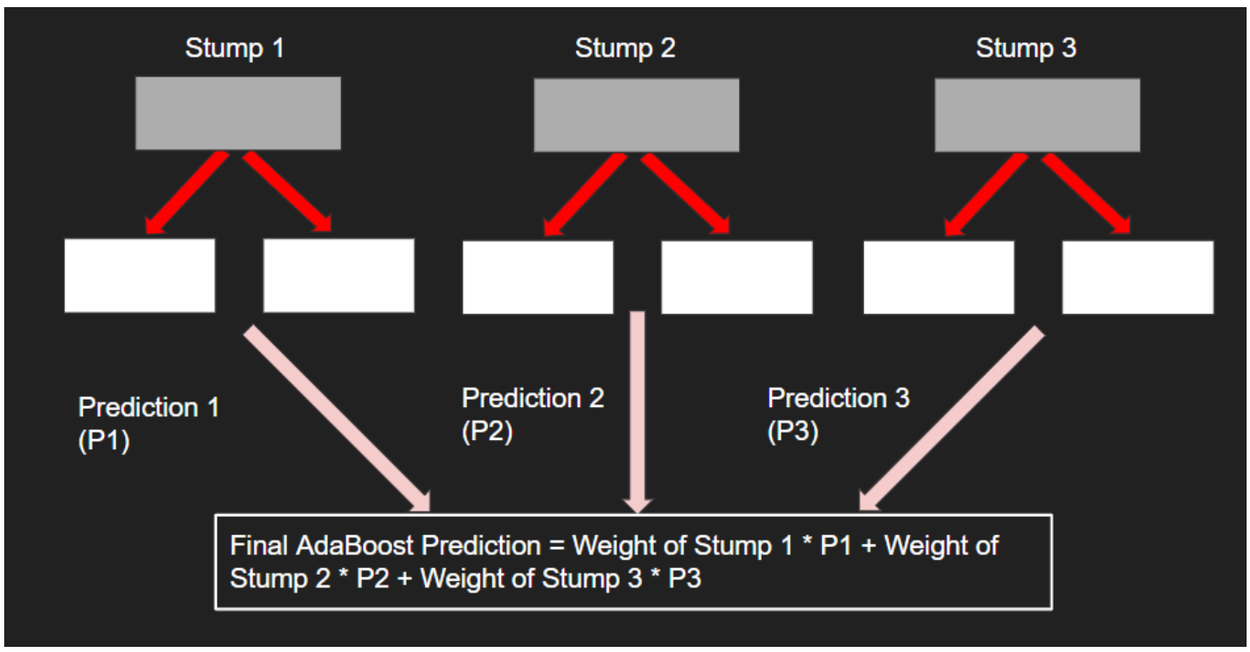
(c) Compute $\epsilon = \frac{\sum_{y_i \neq K_m(x_i)} w_i^{(m)}}{\sum_{y_i} w_i^{(m)}}$, where y_i is the ground truth value of the target variable, and w_i^m is the weight of the sample i at iteration m.

(d) Compute $\alpha_m = \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}$.

(e) Update all the weights $w_i^{(m+1)} = w_i^{(m)} e^{-\alpha_m y K_m(x)}$.

3. New predictions computed by $K(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m K_m(x)\right]$.

- AdaBoost for classification is a supervised machine learning problem.
- It consists of iteratively training multiple stumps using feature data (x) and target labels (y). After training, the model's overall predictions come from a weighted sum of the stumps' predictions.

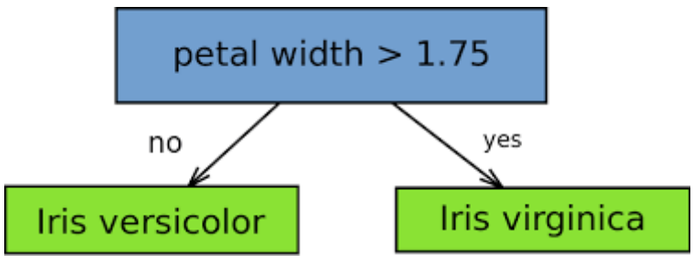


Step 1 - Initialization

- Step 1 of AdaBoost is initializing a weight of 1/N for each datapoint. Again, the weight of a datapoint represent the probability of selecting it during sampling.

Step 2 - Training Weak Classifiers

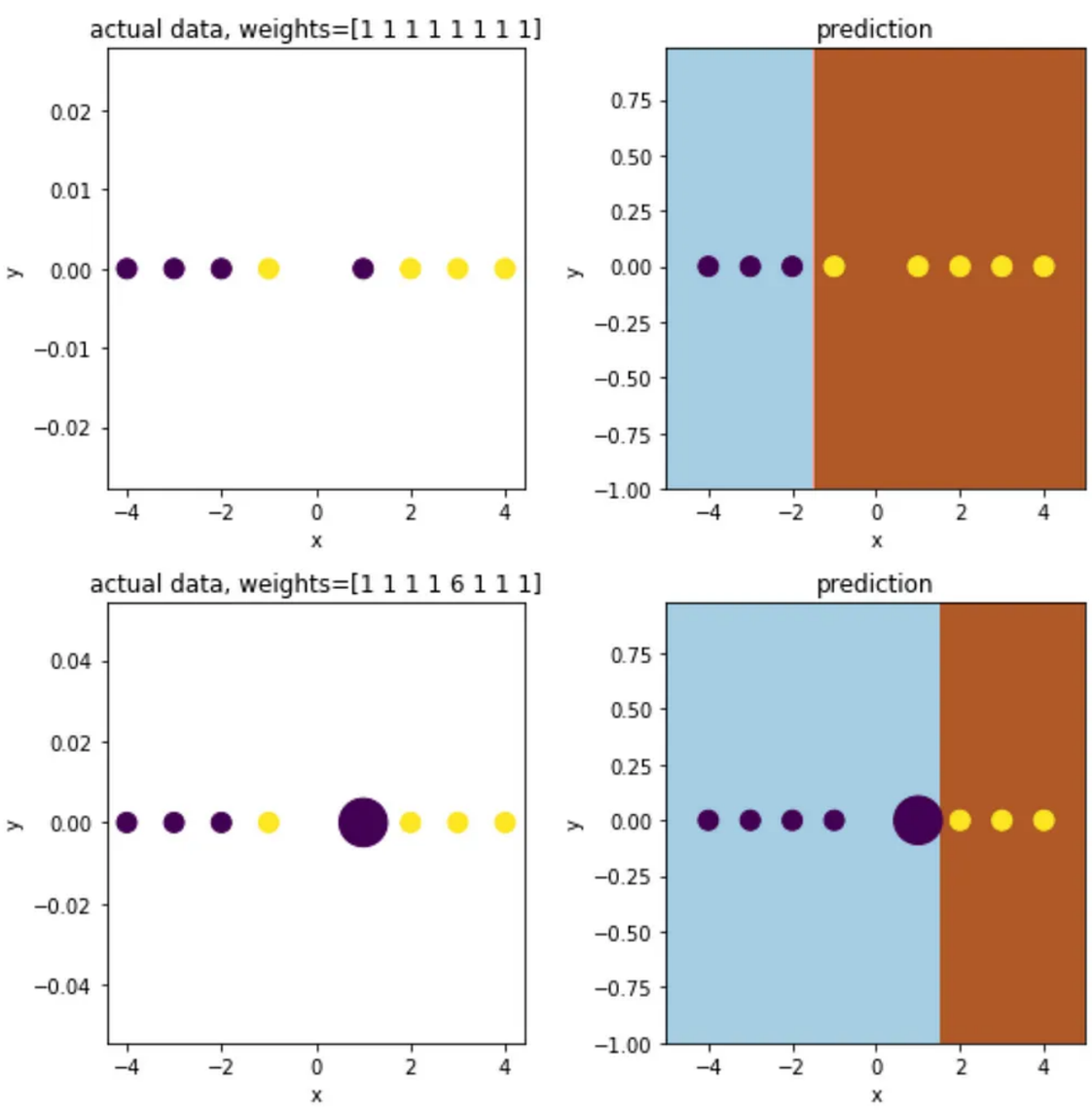
- After sampling the dataset to get the training set for the current iteration, a weak classifier Km is then trained using this training set. In our case, K, is just a decision tree stump.
- Decision tree stump is a decision tree with just one decision. leadina to two or more leaves



Step 3 - Calculating Update Parameters from Error

- Stumps in AdaBoost progressively learn from previous stumps' mistakes through adjusting the weights of the dataset to accommodate the mistakes made by stumps at each iteration.
- The weights of misclassified data will be increased, and weights of correctly classified data will be decreased. As a result, as we go into further iterations, the training data will mostly comprise data that is often misclassified by our stumps.
- Intuitively, if one stump is unable to classify certain data properly, then you just need to train more stumps on the data that is "hard to classify". Hence, taking a weighted sum of these stumps let's AdaBoost perform very well.

Example



- The weight updates at certain iteration m are calculated as the product of the current weights, with the exponential of (- alpha y $K_m(x)$), where

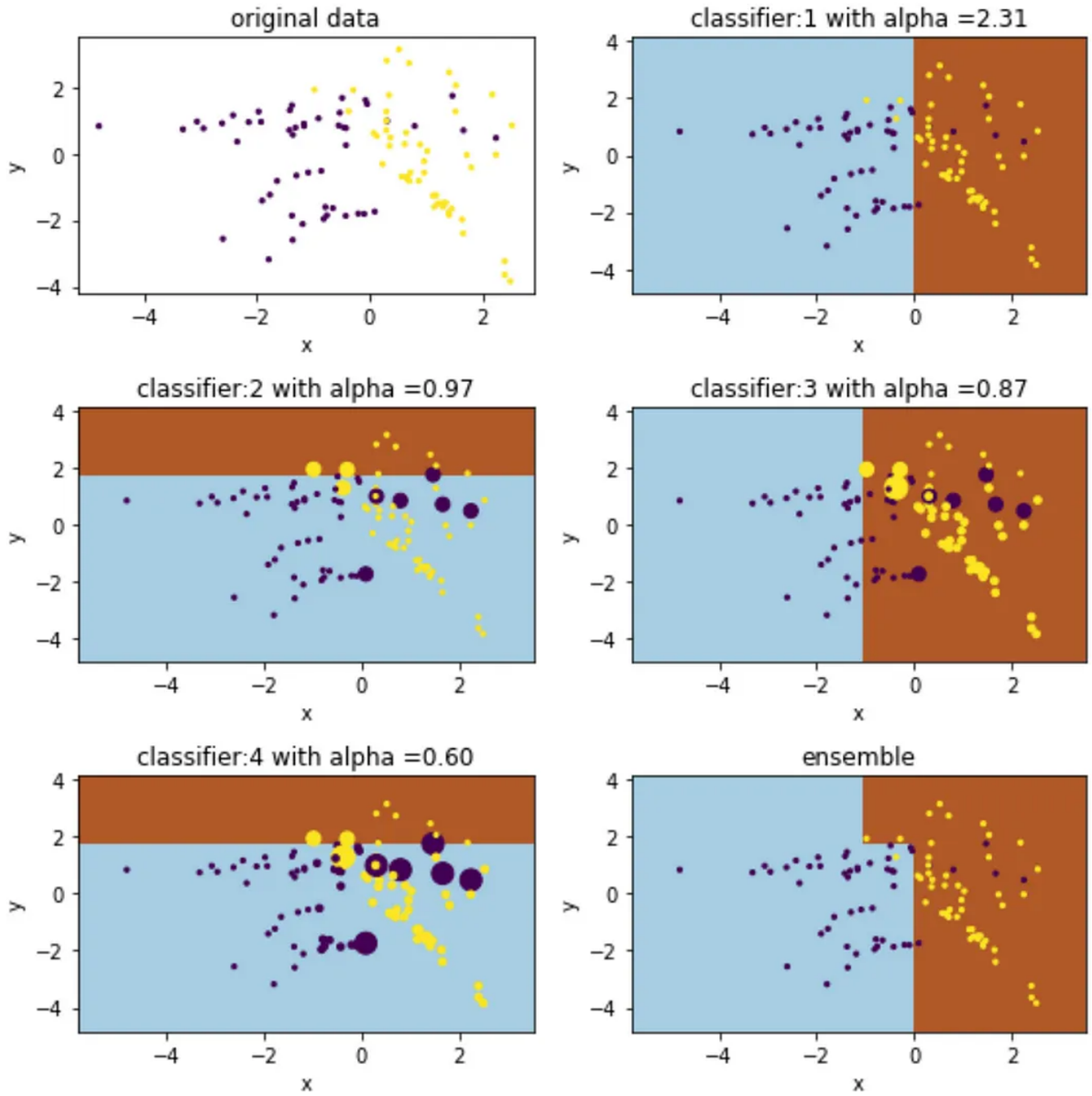
    - y represents the true values of the target feature (in our toy dataset case, this would be the 'species' column).
    - $K_m(x)$ is the prediction made by the stump we trained at iteration m.
    - $alpha_m$ is the confidence we place on the predictive power of stump m.
    - y and $K_m(x)$ can each take values -1 or 1. If y is 1 and $K_m(x)$ is 1, or both are -1, then this means that training sample x has been correctly classified. y * $K_m(x)$ will then be equal to 1.
    - Otherwise, y * $K_m(x)$ will be equal to -1.
    - This corresponds with the "learning from mistakes" concept
    - When data is misclassified, y * $K_m(x)$) = -1, then the weight update will be positive and the weights will exponentially increase for that data.
    - When data is correctly classified, y * $K_m(x)$ = 1, the opposite happens.
- Epsilon is the ratio of sum of weights for misclassified samples to the total sum of weights for samples.
- Large epsilon values, which mean large misclassification percentage, makes alpha_m exponentially decrease.
- The confidence we have in stump m's predictions also exponentially decrease. Intuitively this makes so much sense, because more errors = less confidence.
- Hence stump m's predictions will only have a very small "say" in the overall prediction.
- Naturally, the opposite follows for small epsilon values
- The overall prediction of AdaBoost at iteration m, denoted by

$$C_m(x) = \sum_{i=1}^{m} \alpha_m K_m(x)$$

- AdaBoost trains a sequence of models with augmented sample weights, generating 'confidence' coefficients alpha for individual classifiers based on errors.
- Low errors leads to large alpha, which means higher importance in the voting. • When making overall predictions for AdaBoost, we use alpha, as the weight of each stump's prediction.

$$prediction = alpha_1 * stump_1 + alpha_2 * stump_2 \ldots$$

Example



Step 4 - Making New Overall Predictions

- The final step of the AdaBoost algorithm involves making overall predictions on new data.

$$New\ predictions\ computed\ by\ K(x) = sign[\sum_{m=1}^{M} \alpha_m K_m(x)]$$

- The overall predictions are just the sums of each stump's predictions, weighted by each stump's alpha (confidence that AdaBoost places on each stump's ability to classify the data).
- Since AdaBoost only predicts {-1, 1}. As long as the weighted sum is above 0, then AdaBoost will predict 1.
- The opposite happens, when the weighted sum is below 0 and AdaBoost will predict -1.

Random Forests

- Random forests are among the most widely used machine learning algorithm, probably due to their relatively good performance "out of the box" and ease of use.
- The random forest algorithm can be understood as bagging with decision trees, but instead of growing the decision trees by basing the splitting criterion on the complete feature set, we use random feature subsets.
- While the size of the feature subset to consider at each node is a hyperparameter that we can tune, we can use NumFeatures = $\log_2$ m + 1.

Chapter-4 Support Vector Machines

4.1 Support Vector Machine

Basic Idea

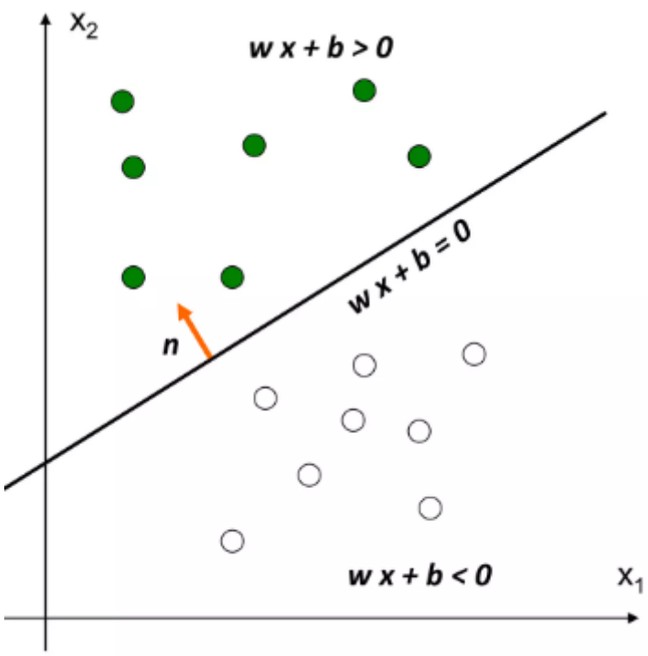
- Optimal hyperplane for linearly separable patterns
- Extend to patterns that are not linearly separable by transformations of original data to map into new space - the Kernel function

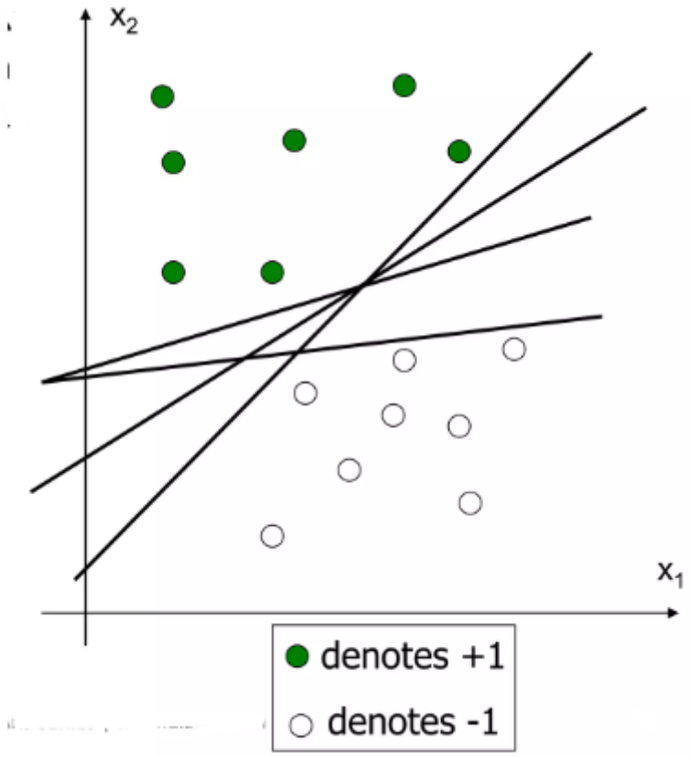
Linear Discriminant Function or a Linear Classifier

- Given the data and two classes, learn a function of the form:

$$g(x) = w^T x + b$$

- A hyper-plane in the feature space
- Decide class = 1 if g(x)>0 and class = -1 otherwise

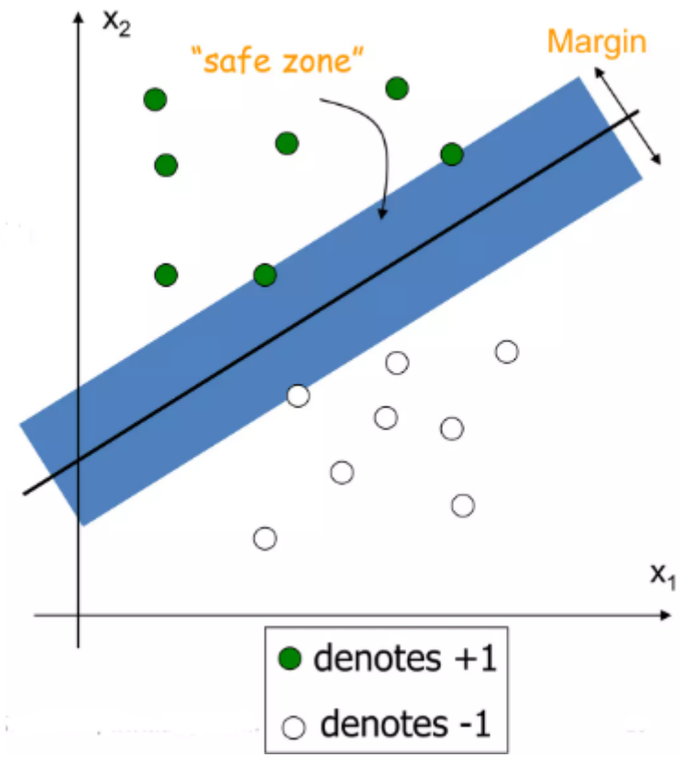




- Infinite number of answers! Which one is the best?

Large Margin Linear Classifier

- The linear discriminant function (classifier) with the maximum margin is the best
- Margin is defined as the width that the boundary could be increased by, before hitting a data point
- Which points should influence optimality?
  - All points?
    - Linear regression
  - Or only "difficult points" close to decision boundary
    - Support vector machines



Support Vectors

- Support vectors are the data points that lie closest to the decision surface (or hyperplane)
- They are the data points most difficult to classify
- They have direct bearing on the optimum location of the decision surface

General input/output for SVMs

- Input: set of (input, output) training pair samples
- Call the input sample features x_1,x_2,...,x_n and the output result y.
- Typically, there can be lots of input features x_i
- Output: set of weights w (or w_i), one for each feature, whose linear combination predicts the value of y.
- Important difference: We use the optimization of maximizing the margin ('street width') to reduce the number of weights that are nonzero to just a few that correspond to the important features that 'matter' in deciding the separating line (hyperplane).
- These nonzero weights correspond to the support vectors (because they 'support' the separating hyperplane)
- Support vectors are the elements of the training set that would change the position of the dividing hyperplane if removed.
- Support vectors are the critical elements of the training set
- The problem of finding the optimal hyper plane is an optimization problem and can be solved by optimization techniques (we use Lagrange multipliers to get this problem into a form that can be solved analytically)

Aim: Learn a large margin classifier

Mathematical Formulation:

$$maximize\ \frac{2}{||w||}$$

such that

$$For\ y_i = +1,\ w^T x_i + b \geq 1$$

$$For\ y_i = -1,\ w^T x_i + b \leq -1$$

$$minimize\ \frac{1}{2}||w||^2$$

such that

$$For\ y_i = +1,\ w^T x_i + b \geq 1$$

$$For\ y_i = -1,\ w^T x_i + b \leq -1$$

- Given a set of data points, define:

$$For\ y_i = +1,\ w^T x_i + b \geq 1$$

$$For\ y_i = -1,\ w^T x_i + b \leq -1$$

Algebraic Expression for Width of a Margin

- Given 2 parallel lines with equations

  $$ax + by + c_1 = 0\ and\ ax + by + c_2 = 0$$

  the distance between them is given by:

$$d = \frac{|c_2 - c_1|}{\sqrt{a^2 + b^2}}$$

Our lines in 2-D are:

$$w_1x_1 + w_2x_2 + b - 1 = 0 \; and \; w_1x_1 + w_2x_2 + b + 1 = 0$$

$$Distance = \frac{|b - 1 - b - 1|}{\sqrt{w_1^2 + w_2^2}} = \frac{2}{||w||}$$

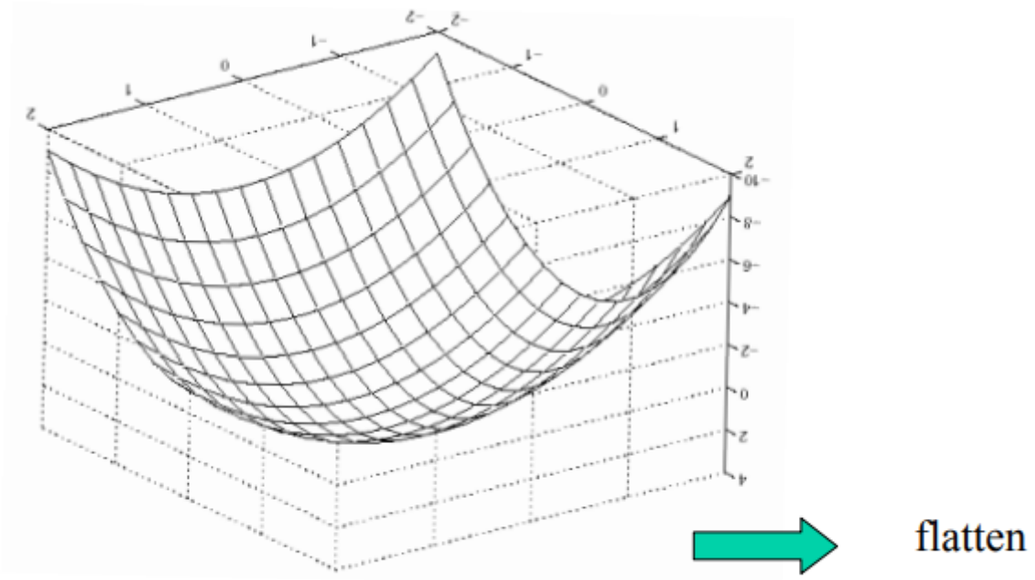
- **Formulation:**

$$minimize \; \frac{1}{2}||w||^2$$

such that

$$y_i(w^T x_i + b) \geq 1$$

- This is a quadratic programming problem with linear constraints, the surface is a paraboloid, with just a single global minimum
- However, we will convert it to Lagrangian dual in order to use the kernel trick!



Example: paraboloid $2+x^2+2y^2$ s.t. $x+y=1$

- Intuition: Find intersection of two functions f, g at a tangent point (intersection = both constraints satisfied; tangent = derivative is 0); this will be a min (or max) for f s.t. the contraint is satisfied
- Rewriting the conditions as:

- Want to look for solution point p where

$$\nabla f(p) = \nabla \lambda g(p)$$

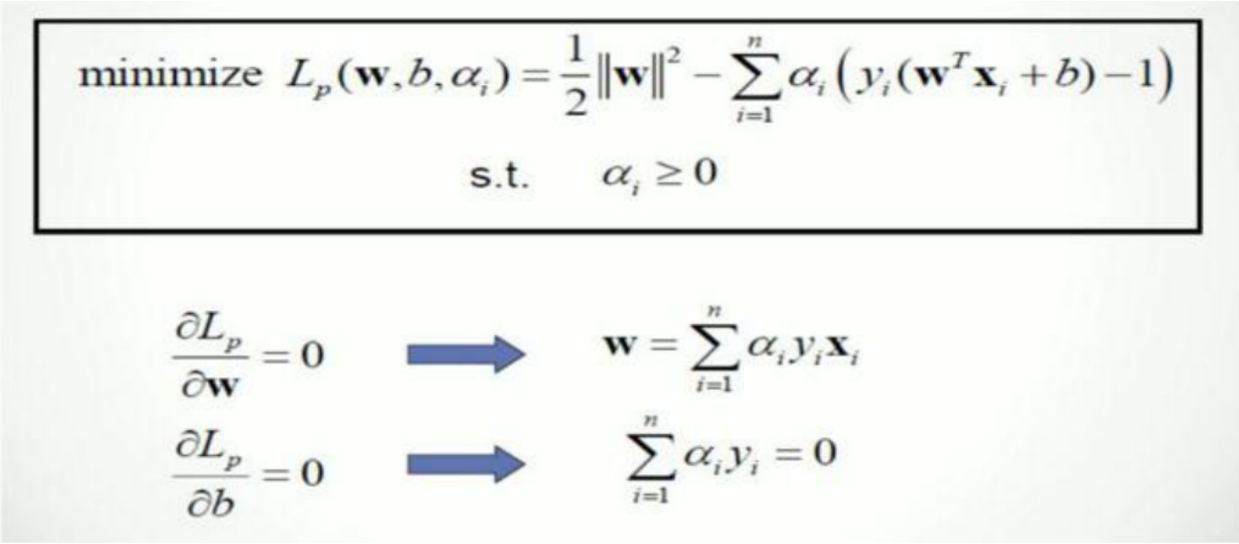
$$g(x) = 0$$

- Or, combining these two as the *Langrangian* L & requiring derivative of L be zero:

$$L(x,a) = f(x) - ag(x)$$

$$\nabla(x,a) = 0$$

Solving the Optimization Problem



- This indicates the primal form of the optimization problem
- We will actually solve the optimization problem by now solving for the dual of this original problem
- **The Lagrangian Dual Problem**: instead of minimizing over w, b, subject to constraints involving a's, we can maximize over a (the dual variable) subject to the relations obtained previously for w and b
- This is a constrained optimization problem.
  - Optimization — because, we are to find the line from which the support vectors are maximally separated and
  - Constrained — because, the support vectors should be away from the road and not on the road. We will use Lagrange Multipliers to solve this problem

4.2 SVM Lagrange Problem

- Lagrange stated that if we want to find the minimum of f under the equality constraint g, we need to solve for:

$$\forall f(x) - \alpha \nabla g(x) = 0$$

  α is called the Lagrange multiplier
- In terms of the SVM optimization problem,

$$f(w) = \frac{1}{2}||w||^2$$

$$g(w,b) = y_i(wx+b) - 1,\ i = 1, \ldots, m$$

- The Lagrangian function is then

$$L(w,b,\alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{m}\alpha_i[y_i(wx+b)-1]$$

Quadratic programming with linear constraints

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

Lagrangian Function

$$\text{minimize} \quad L_p(\mathbf{w}, b, \alpha_i) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 \right)$$

$$\text{s.t.} \quad \alpha_i \geq 0$$

Primal problem:

$$\min L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} a_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) + \sum_{i=1}^{l} a_i$$

$$\text{s.t. } \forall i \; a_i \geq 0$$

$$\mathbf{w} = \sum_{i=1}^{l} a_i y_i \mathbf{x}_i, \quad \sum_{i=1}^{l} a_i y_i = 0$$

Dual problem:

$$\max L_D(a_i) = \sum_{i=1}^{l} a_i - \frac{1}{2} \sum_{i=1}^{l} a_i a_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j \right)$$

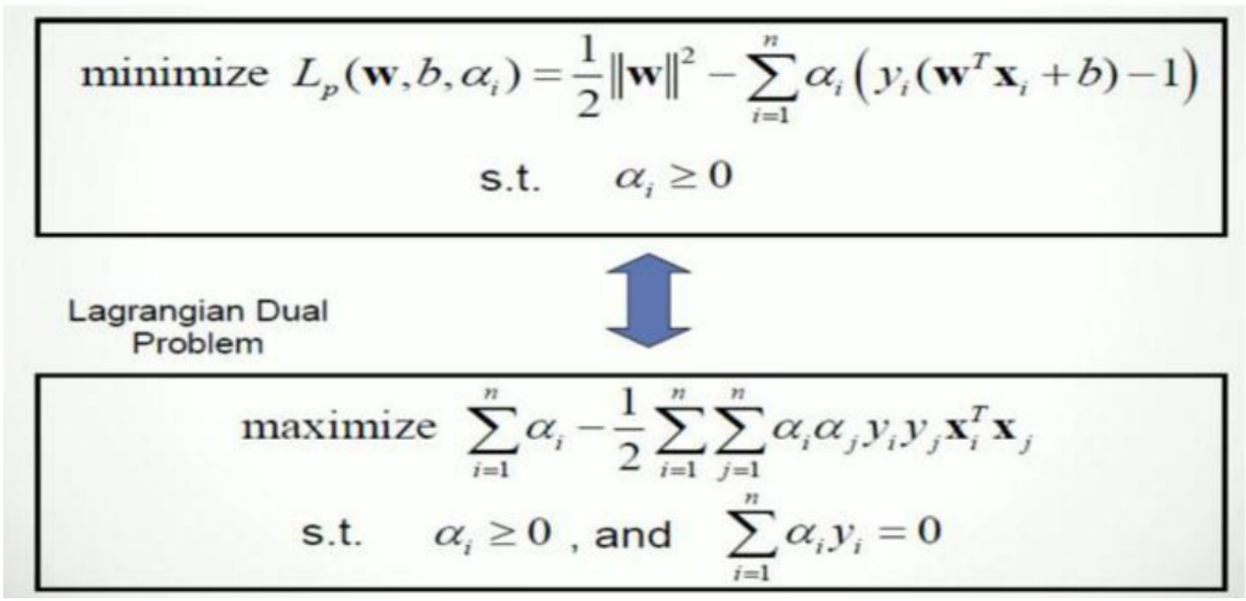
$$\text{s.t.} \quad \sum_{i=1}^{l} a_i y_i = 0 \;\&\; a_i \geq 0$$

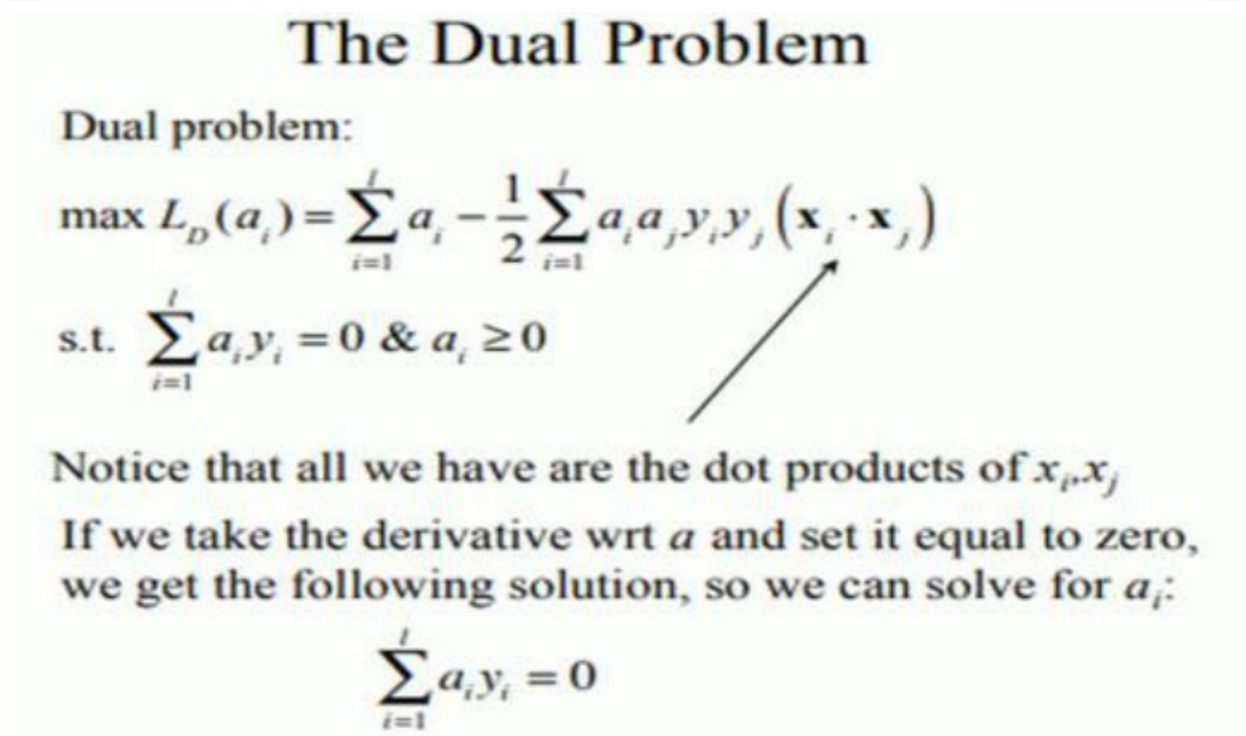
(note that we have removed the dependence on **w** and b)

The Dual problem

- For any convex optimization problem, there always exist settings of the dual variables such that the unconstrained minimum of the Lagrangian with respect to the primal variables (keeping the dual variables fixed) coincides with the solution of the original constrained minimization problem.
- Kuhn-Tucker theorem: The solution we find here will be the same as the solution to the original problem
- The advantage of the dual problem over the Lagrange primal problem is that the objective function now only depends on the Lagrangian multipliers, which is easier to be solved analytically.
- Also it will let us solve the problem by computing the just the inner products of x_i, x_j

Solving the Optimization Problem





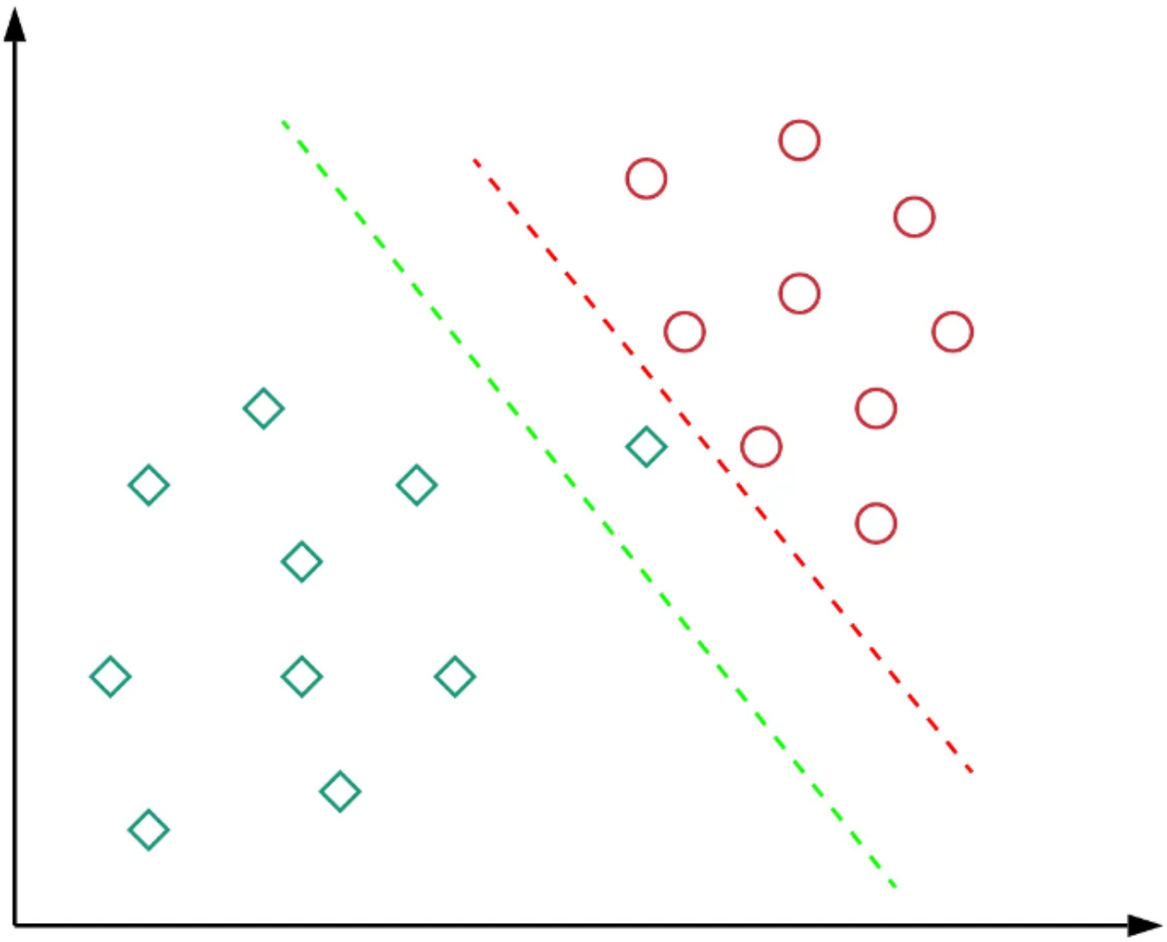
- The linear discriminant function is:

$$g(x)=w^Tx+b=\sum_{i\in SV}\alpha_i x_i^T x+b$$

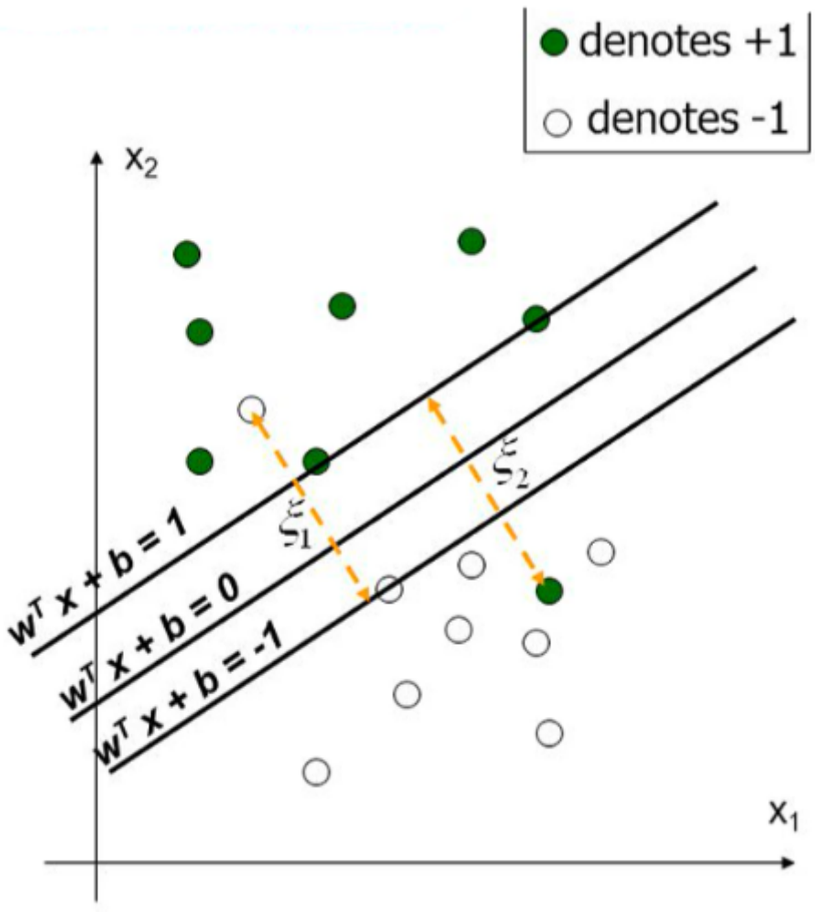
- Notice it relies on a dot product between the test point x and the support vectors x_i
- Also keep in mind that solving the optimization problem involved computing dot products $x_i^T x_j$between all pairs of training points

Soft Margin Formulation

This idea is based on a simple premise: allow SVM to make a certain number of mistakes and keep margin as wide as possible, so that other points can still be classified correctly. This can be done simply by modifying the objective of SVM.



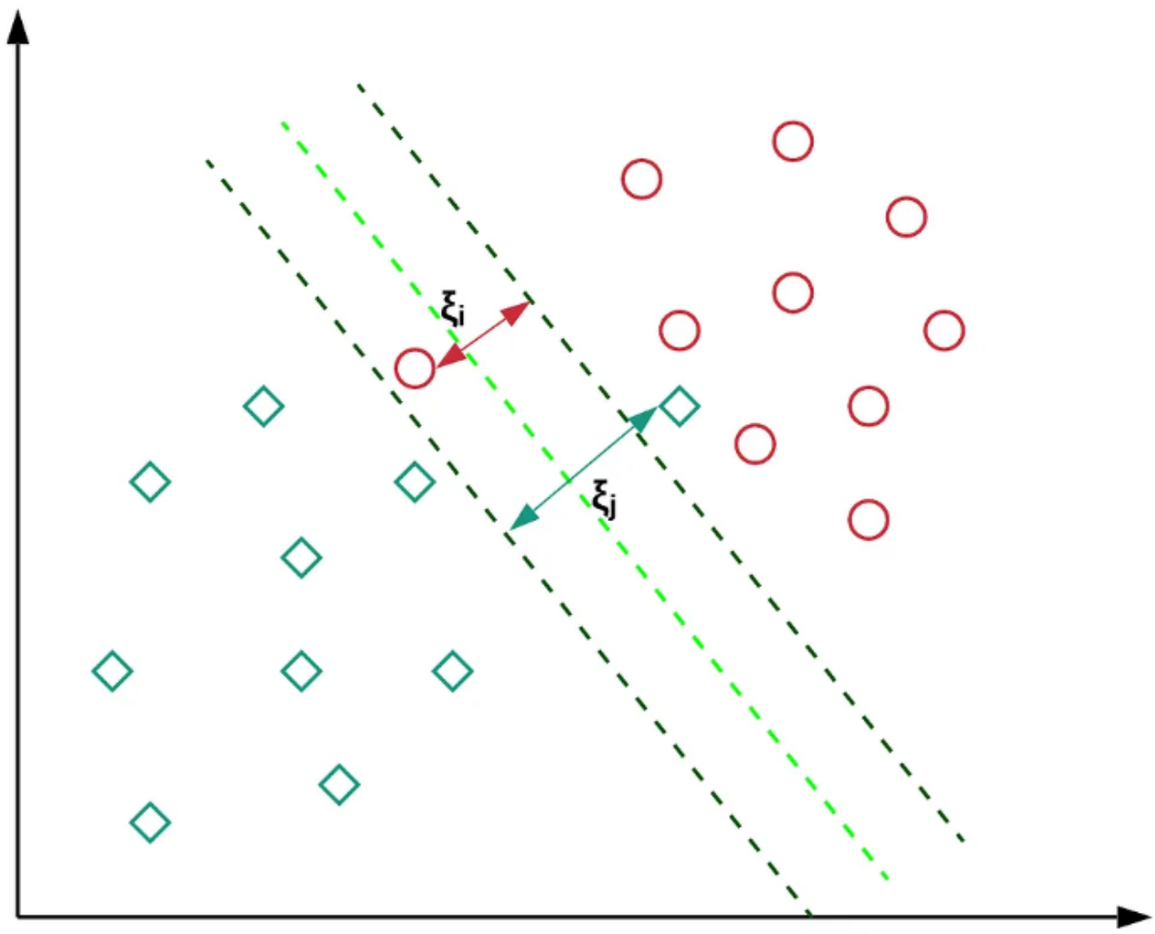
- Here the red decision boundary perfectly separates all the training points.
- The green decision boundary has a wider margin that would allow it to generalize well on unseen data.
- In that sense, soft margin formulation would also help in avoiding the overfitting problem.
- So in such cases, where the data is not linearly separable (noisy data, outliers, etc.)
- Slack variables ξ_i can be added to allow misclassification of difficult or noisy data points



- In the problem posed here, we soften the constraint:

$$y_i(x_i^T w + b) \geq 1 - \xi_i,\ \xi_i \geq 0$$

- The new constraint permits a functional margin that is less than 1
- We thus state a preference for margins that classify the training data correctly, but soften the constraints to allow for non-separable data with a penalty proportional to the amount by which the example is misclassified.



- The objective contains a penalty of cost $C\xi_i$, for any data point that
  - falls within the margin on the correct side of the separating hyperplane (i.e., when $0 < \xi_i \leq 1$)
  - or on the wrong side of the separating hyperplane (i.e., when $\xi_i > 1$)
  - The value of ξ_i is the distance of x_i from the corresponding class's margin if x_i is on the wrong side of the margin, otherwise zero.
  - Thus the points that are far away from the margin on the wrong side would get more penalty.
- C is a hyperparameter that decides the trade-off between maximizing the margin and minimizing the mistakes. When C is small, classification mistakes are given less importance and focus is more on maximizing the margin and vice versa.
- We would therefore like to minimize the sum total of the penalties ξ_i over all i (this is an upper bound on the training classification errors). Thus, the new well-posed optimization problem (using L_1 regularization) becomes:

$$min\ \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i$$

$$s.t.\ y_i(x_i^T w + b) \geq 1 - \xi_i,\ \xi_i \geq 0$$

- Formulation:

$$\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

$$\text{minimize } \frac{1}{2}\|\mathbf{w}\|^2$$

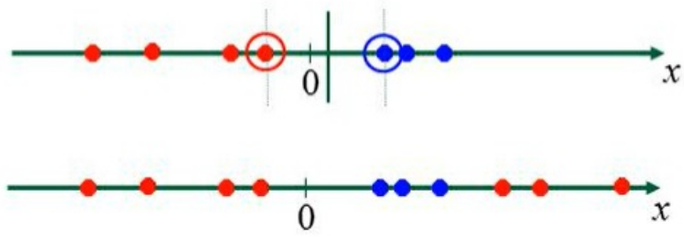
$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

Without slack variables

- Parameter C can be viewed as a way to control overfitting

4.3 Non Linear SVM

- Datasets that are linearly separable with noise work out great
- But what are we going to do if the dataset is just too hard?
- Kernel Trick!!!
  - SVM = Linear SVM + Kernel Trick

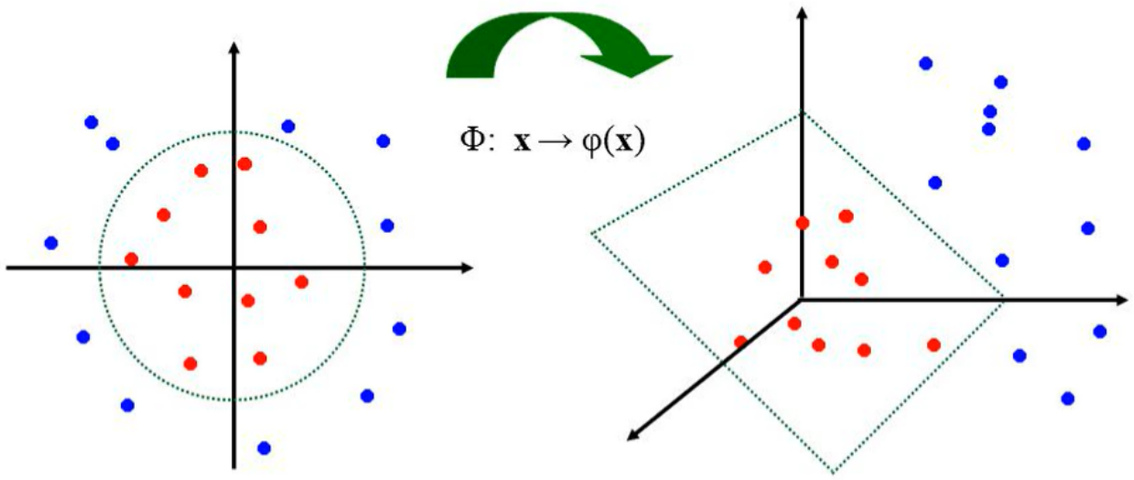


Kernel Trick Motivation

- Linear classifiers are well understood, widely-used and efficient.
- How to use linear classifiers to build non-linear ones?
- Neural networks: Construct non-linear classifiers by using a network of linear classifiers (perceptrons).
- Kernels:
  - Map the problem from the input space to a new higher-dimensional space (called the feature space) by doing a non-linear transformation using a special function called the kernel.
  - Then use a linear model in this new high-dimensional feature space.
  - The linear model in the feature space corresponds to a non-linear model in the input space.

Feature Space

- General idea: the original input space can be mapped to some higher-dimensional feature space where the training set is separable:



- Proper feature mapping can make non-linear to linear!

The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(x) = w^T\phi(x) + b = \sum_{i \in SV} \alpha\phi(x_i)^T\phi(x) + b$$

- No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.
- A kernel function is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(x_i, x_j) \equiv \phi(x_i)^T\phi(x_j)$$

Solving the Optimization Problem

- One particularly interesting consequence of strong duality for convex optimization problems is a property known as complementary slackness
- According to the complementary slackness condition:

$$\alpha_i(y_i(w^Tx_i + b) - 1) = 0$$

- Thus, only support vectors have $\alpha_i \neq 0$
- The solution has the form:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i = \sum_{i \in SV} \alpha_i y_i x_i$$

  get b from $y_i(w^Tx_i+b) - 1 = 0$, where x_i is support vector
- Now knowing the α_i's we can find the weights w for the maximal margin separating hyperplane:

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i$$

- and now, after training and finding the w by this method, given an unknown point u measured on features x_i we can classify it by looking at the sign of:

$$f(x) = wu + b = (\sum_{i=1}^{l} \alpha_i y_i x_i u) + b$$

Simple Example

- x = (x1, x2, x3); y = (y1, y2, y3).

Then for the function f(x) = (x1x1, x1x2, x1x3, x2x1, x2x2, x2x3, x3x1, x3x2, x3x3)

The kernel is K(x, y) = (<x, y>)^2.

- Let's plug in some numbers to make this more intuitive:

Suppose x = (1, 2, 3); y = (4, 5, 6). Then:
f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)
f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)
<f(x), f(y)> = 16 + 40 + 72 + 40 + 100+ 180 + 72 + 180 + 324 = 1024

A lot of algebra. Mainly because f is a mapping from 3-dimensional to 9 dimensional space.

- Now let us use the kernel instead:
  K(x, y) = (4 + 10 + 18) ^2 = 32^2 = 1024
  Same result, but this calculation is so much easier.

Another Example

2-dimensional vectors $\mathbf{x}=[x_1 \; x_2]$;

let $K(\mathbf{x}_i,\mathbf{x}_j)=(1+\mathbf{x}_i^T\mathbf{x}_j)^2$,

Need to show that $K(\mathbf{x}_i,\mathbf{x}_j)=\varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j)$:

$$K(\mathbf{x}_i,\mathbf{x}_j)=(1+\mathbf{x}_i^T\mathbf{x}_j)^2,$$
$$= 1+x_{i1}^2x_{j1}^2+2\,x_{i1}x_{j1}x_{i2}x_{j2}+x_{i2}^2x_{j2}^2+2x_{i1}x_{j1}+2x_{i2}x_{j2}$$
$$= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\,x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}x_{i1} \;\; \sqrt{2}x_{i2}]^T [1 \;\; x_{j1}^2 \;\; \sqrt{2}\,x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}]$$
$$= \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j), \quad \text{where } \varphi(\mathbf{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\,x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$$

Optimization

- Formulation: (Lagrangian Dual Problem)

$$\text{maximize} \quad \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$$

such that

$$0 \le \alpha_i \le C$$
$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

- The solution of the discriminant function is

$$g(\mathbf{x}) = \sum_{i\in SV}\alpha_i K(\mathbf{x}_i,\mathbf{x}) + b$$

- The optimization technique is the same.

Examples of Commonly used Kernel Functions

- Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
- Gaussian (Radial-Basis Function (RBF)) kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}\right)$$

- Sigmoid:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$$

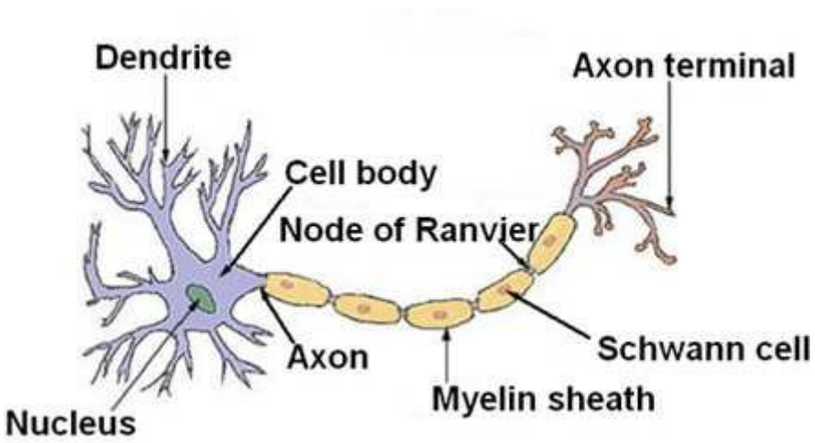
Algorithm

- Choose a kernel function
- Choose a value for C
- Solve the quadratic programming problem
- Construct the discriminant function from the support vectors

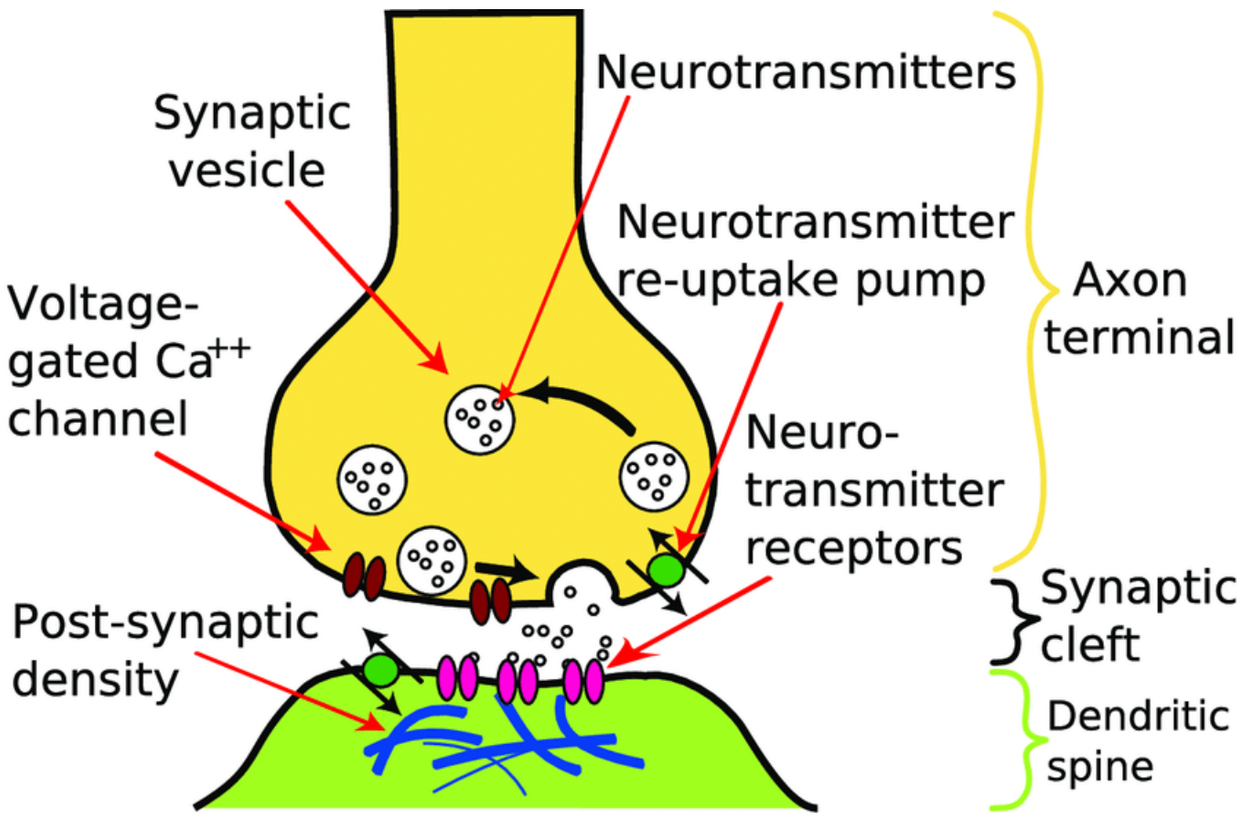
Chapter-5 Artificial Neural Networks

5.1 Perceptron

- Neural networks, a biologically-inspired programming paradigm which enables a computer to learn from observational data
- Neural networks arise from attempts to model human/animal brains
- We will focus on multi-layer perceptrons
- The human brain is made up of about 100 billion neurons
- Neurons receive electric signals at the dendrites and send them to the axon

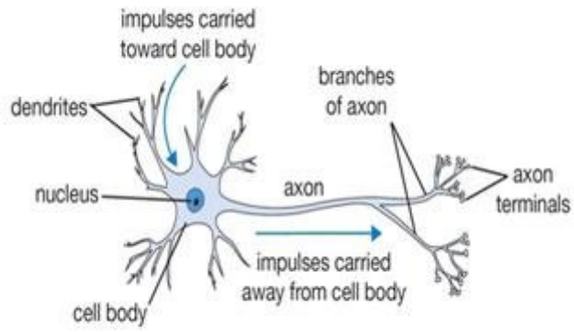


- The axon of the neuron is connected to the dendrites of many other neurons

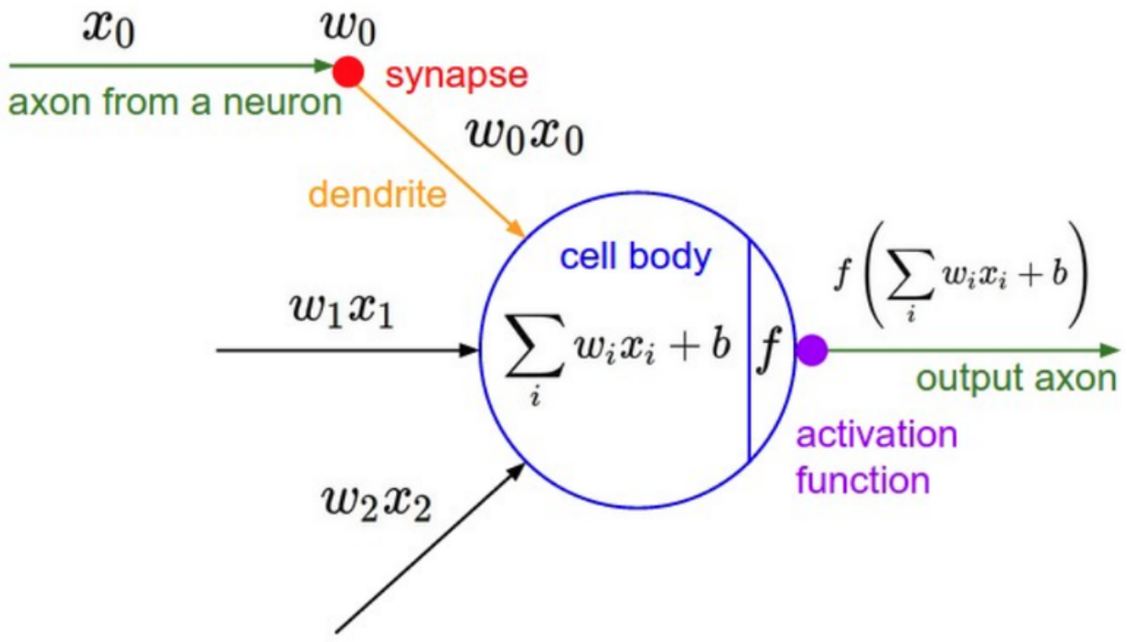


- The human brain consists primarily of nerve cells called neurons, linked together with other neurons via strands of fiber called axons.
- Axons are used to transmit nerve impulses from one neuron to another whenever the neurons are stimulated. A neuron is connected to the axons of other neurons via dendrites, which are extensions from the cell body of the neuron.
- The contact point between a dendrite and an axon is called a synapse.

- Neurologists have discovered that the human brain learns by changing the strength of the synaptic connection between neurons upon repeated stimulation by the same impulse.



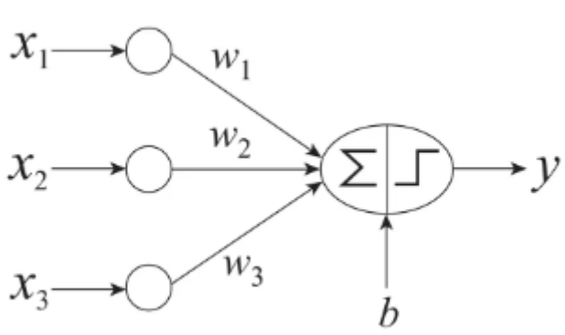
Artificial neuron - biological motivation



Brain vs. Artificial Neural Networks

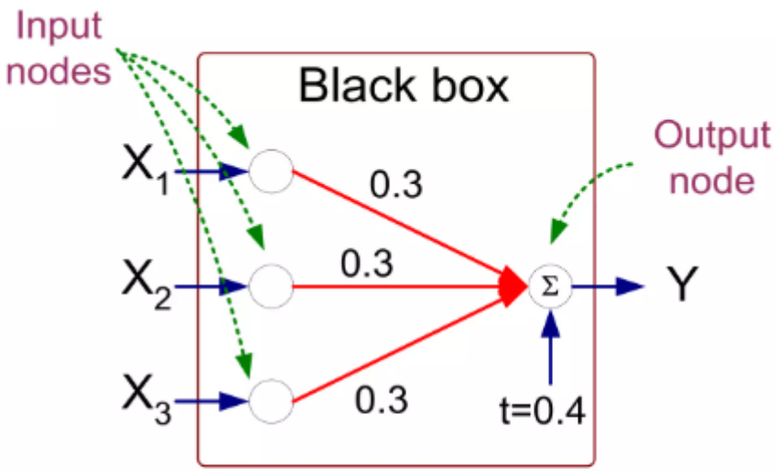
- Similarities
  - Neurons, connections between neurons
  - Learning = change of connections, not change of neurons
  - Massive parallel processing
- But artificial neural networks are much simpler
  - computation within neuron vastly simplified
  - discrete time steps
  - typically some form of supervised learning with massive number of stimuli

Basic Architecture of Perceptron



Perceptron Example

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Learning Perceptron Model

- In a perceptron, each input node is connected via a weighted link to the output node.
- The weighted link is used to emulate the strength of synaptic connection between neurons.
- As in biological neural systems, training a perceptron model amounts to adapting the weights of the links until they fit the input-output relationships of the underlying data.
- During the training phase of a perceptron model, the weight parameters w are adjusted until the outputs of the perceptron become consistent with the true outputs of training examples.

Perceptron Learning Rule

- Initialize the weights (w_0, w_1, …, w_d)
- Repeat
  - For each training example (x_i, y_i)
    - Compute $\hat{y}_i$
    - Update the weights:

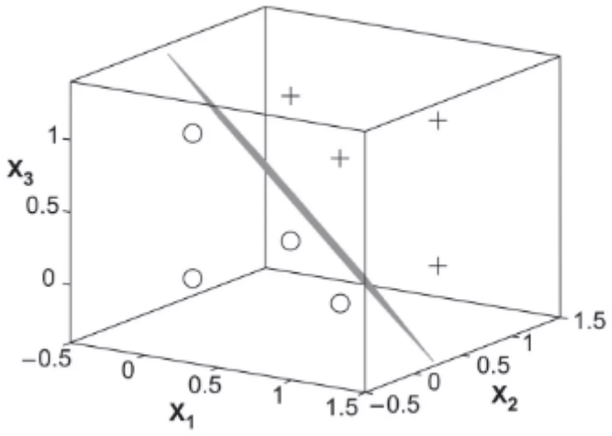
$$w_j^{(k+1)} = w_j^{(k)} + \lambda\left(y_i - \hat{y}_i^{(k)}\right)x_{ij}$$

- Until stopping condition is met
- k: iteration number; λ: learning rate

- Weight update formula:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda\left(y_i - \hat{y}_i^{(k)}\right)x_{ij}$$

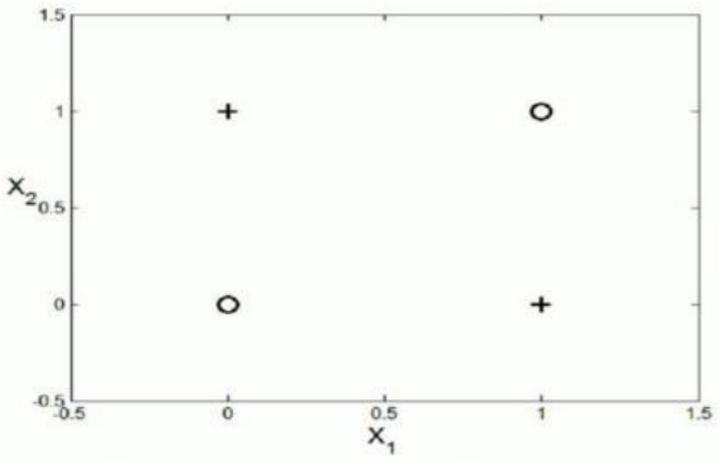
- Intuition:
  - Update weight based on error: e = $(y_i - \hat{y}_i)$
    - If y = $\hat{y}$, e=0: no update needed
    - If y > $\hat{y}$, e=2: weight must be increased (assuming Xij is positive) so that $\hat{y}$ will increase
    - If y < $\hat{y}$, e=-2: weight must be decreased (assuming Xij is positive) so that $\hat{y}$ will decrease



- Since y is a linear combination of input variables, decision boundary is linear 0

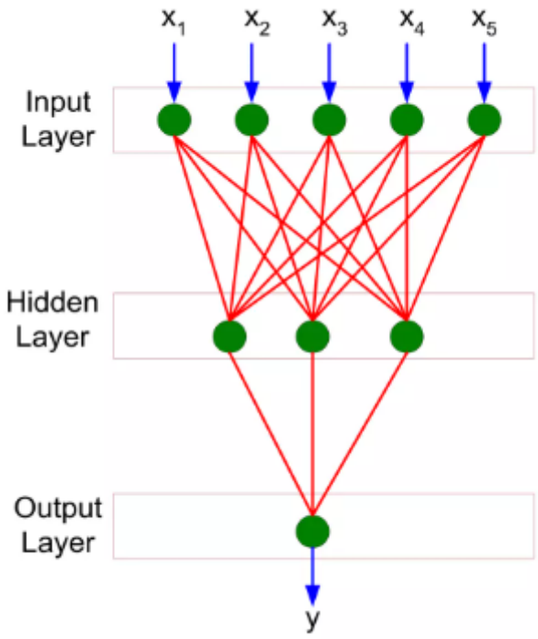
Nonlinearly Separable Data

x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1

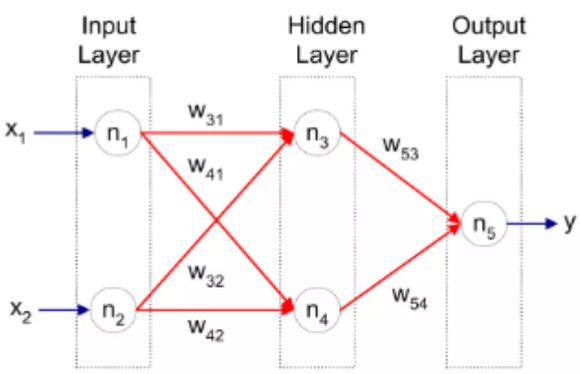


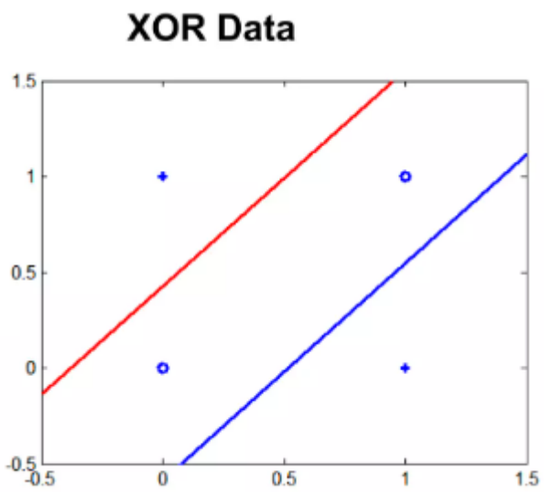
- For a XOR classification problem, there is no linear hyperplane that can separate the two classes.

Multi-layer Neural Network



- An ANN has a more complex structure than that of a perceptron model
- More than one hidden layer of computing nodes
- Every node in a hidden layer operates on activations from preceding layer and transmits activations forward to nodes of next layer
- Also referred to as "feedforward neural networks"





- Multi-layer neural networks are able to model more complex relationships between input and output variables
- We can think of each hidden node as a perceptron that tries to construct one of the two hyperplanes, while the output node simply combines the results of the perceptrons to yield the decision boundary
- Our goal is to extend this model by making the basis functions $\varphi_j(x)$ depend on parameters and then to allow these parameters to be adjusted, along with the coefficients w_j during training.
- Neural networks use basis functions such that each basis function is itself a nonlinear function of a linear combination of the inputs, where the coefficients in the linear combination are adaptive parameters.

5.2 Feed-Forward Networks

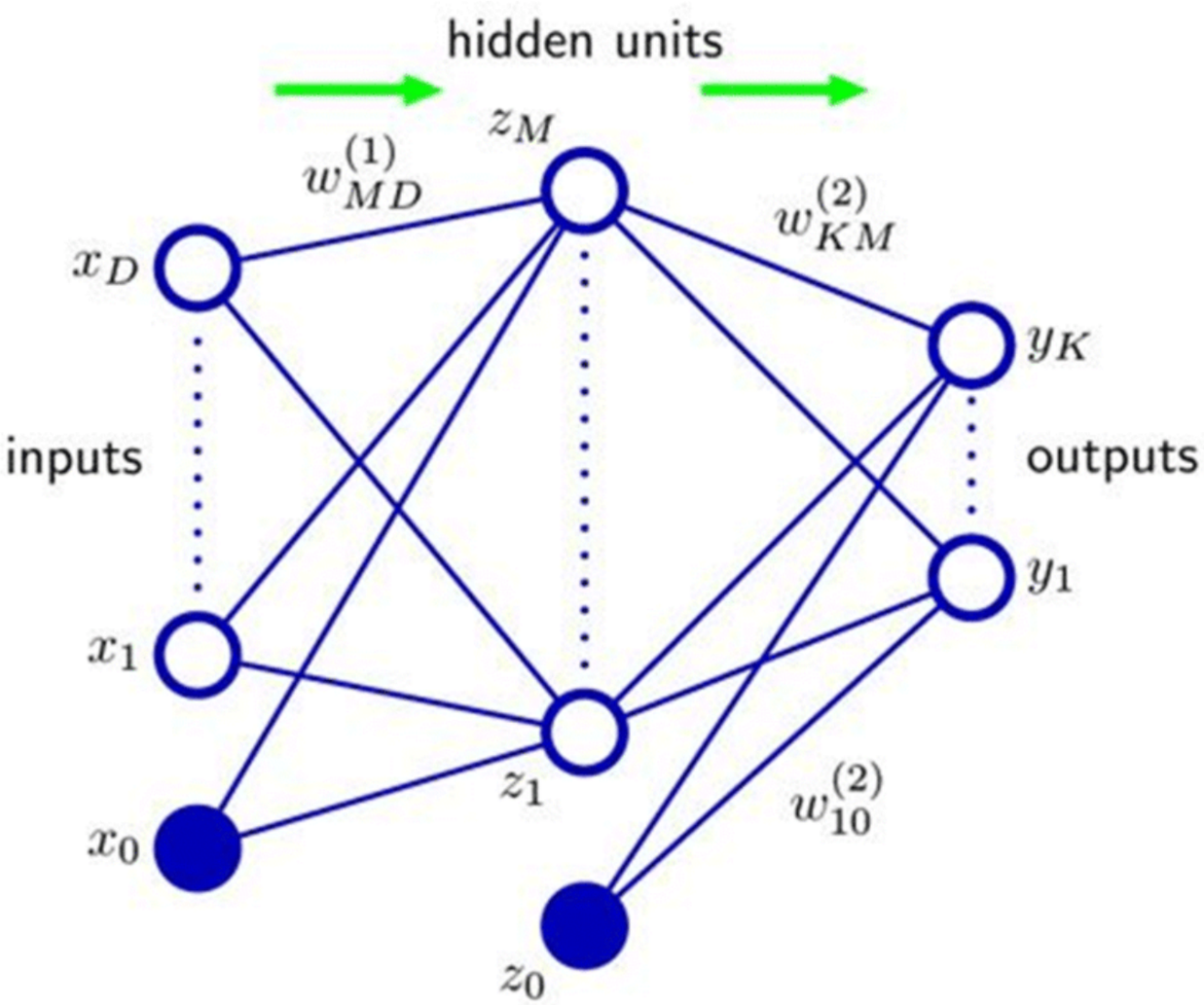
- We have looked at generalized linear models of the form:

$$y(x, w) = f\left(\sum_{j=1}^{M} w_j \phi_j(x)\right)$$

- for fixed non-linear basis functions φ(.) where f( ) is a nonlinear activation function in the case of classification.
- We now extend this model by allowing adaptive basis functions, and learning their parameters.
- In feed-forward networks (a.k.a. multi-layer perceptrons) we let each basis function be another non-linear function of linear combination of the inputs.

$$\phi_j(x) = f\left(\sum_{j=1}^{M} \dots\right)$$

Network diagram for a Two Layer Neural Network



- Connect together a number of units into a feed-forward network(DAG)
- Above shows a network with one layer of hidden units
- Starting with input x = (x_1, ..., x_D), construct M linear combinations:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

- where j = 1, ..., M and the superscript (1) indicates that the corresponding parameters are in the first 'layer' of the network. These a_j's are known as activations.
  - Each of them is then transformed using a differentiable, nonlinear activation function h(.) to give

$$z_j = h(a_j)$$

- The nonlinear functions h(.) are generally chosen to be sigmoidal functions such as the logistic sigmoid or the 'tanh' function.
- These values are again linearly combined to give output unit activations

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}$$

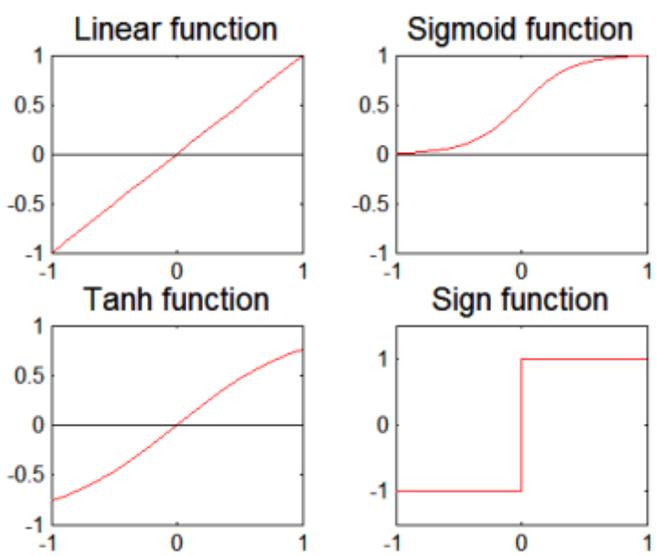
- where k = 1, ..., K and K is the total number of outputs.
- This transformation corresponds to the second layer of the network, and again the $w_{k0}^{(2)}$ are bias parameters.
- Finally, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k
- We can combine these various stages to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$y_k(x, w) = \sigma \left(\sum_{j=1}^{M} w_{kj}^{(2)} h \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

- Thus the neural network model is simply a nonlinear function from a set of input variables x_i to a set of output variables y_k controlled by a vector w of adjustable parameters.

Different Activation Functions

- Activation function of a node in an artificial neural network is a function that calculates the output of the node (based on its inputs and the weights on individual inputs).



Parameter Optimization

- We turn next to the task of finding a weight vector w which minimizes the chosen function E(w).
- Because the error E(w) is a smooth continuous function of w, its smallest value will occur at a point in weight space such that the gradient of the error function vanishes, so that ∇E(w) = 0 as otherwise we could make a small step in the direction of -∇E(w) and thereby further reduce the error.
- Points at which the gradient vanishes are called stationary points, and may be further classified into minima, maxima, and saddle points. lead
- A minimum that corresponds to the smallest value of the error function for any weight vector is said to be a global minimum.

- Any other minima corresponding to higher values of the error function are said to be local minima.
- For a successful application of neural networks, it may not be necessary to find the global minimum (and in general it will not be known whether the global minimum has been found) but it may be necessary to compare several local minima in order to find a sufficiently good

5.3 Error Backpropagation

- The goal is to find an efficient technique for evaluating the gradient of an error function E(w) for a feed-forward neural network. This can be achieved using a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as error backpropagation, or sometimes simply as backprop.

Network Training

- Given a specified network structure, how do we set its parameters (weights)?
- As usual, we define a criterion to measure how well our network performs, optimize against it.
- For regression, training data comprising a set of input vectors $\{x_n\}$, where n = 1 ,..., N, together with a corresponding set of target vectors $\{t_n\}$

$$E(w) = \sum_{n=1}^{N} \{y(x_n, w) - t_n\}^2$$

- For binary classification, we consider a discriminative model, where conditional distribution of targets given inputs is a Bernoulli distribution of the form:

$$p(t|w) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n}$$

$$E(w) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

- We now derive the backpropagation algorithm for a general network having arbitrary feed-forward topology, arbitrary differentiable nonlinear activation functions, and a broad class of error function.
- The sum is transformed by a nonlinear activation function h(.) to give the activation z_j of unit j in the form $z_j = h(a_j)$
- For each pattern in the training set, we shall suppose that we have supplied the corresponding input vector to the network and calculated the activations of all of the hidden and output units in the network by successive application of the two equations.
- This process is often called forward propagation because it can be regarded as a forward flow of information through the network.
- We shall now see how this simple result extends to the more complex setting of multilayer feed-forward networks.
- In a general feed-forward network, each unit computes a weighted sum of its inputs of the form

$$a_j = \sum_i w_{ji} z_i$$

- where z, is the activation of a unit, or input, that sends a connection to unit j, and w_{ji} is the weight associated with that connection.
- We now introduce a useful notation, where the δ's are often referred to as errors

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

- We can write

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \; because \; a_j = \sum_i w_{ji} z_i$$

- and then we can obtain

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

- The required derivative is obtained simply by multiplying the value of δ for the unit at the output end of the weight by the value of z for the unit at the input end of the weight.
- This takes the same form as for the simple linear model.
- Now consider the evaluation of the derivative of E_n with respect to a weight w_{ji}
- The outputs of the various units will depend on the particular input pattern n.
- First we note that E_n depends on the weight w_{ji} only via the summed input a_j to unit j.
- We can therefore apply the chain rule for partial derivatives to give

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

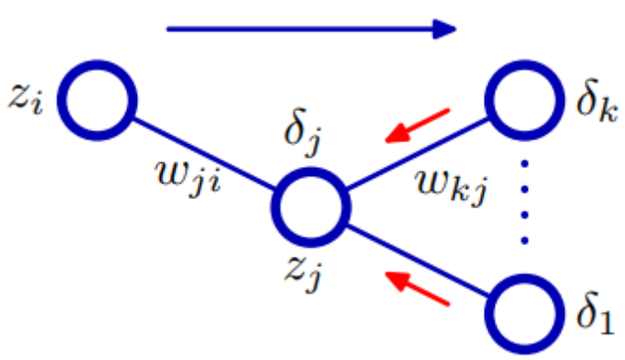
- As we have seen already, for the output units, we have

$$\delta_k = y_k - t_k$$

- To evaluate the δ's for hidden units, we again make use of the chain rule for partial derivatives, where the sum runs over all units k to which unit j sends connections.

$$\delta_j = \frac{\partial E_n}{\partial a_j} \equiv \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

- Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections.
- The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.



- To evaluate the δ's for hidden units, we again make use of the chain rule for partial derivatives, where the sum runs over all units k to which unit j sends connections.
- We obtain the following backpropagation formula which tells us that the value of δ for a particular hidden unit can be obtained by propagating the δ's backwards from units higher up in the network

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k}$$

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

$$a_k = \sum_i w_{ki} z_i = \sum_i = w_{ki} h(a_i)$$

$$\frac{\partial a_k}{\partial a_j} = \sum_k = w_{kj} h'(a_j)$$

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

- Apply an input vector x_n to the network and forward propagate through the network using

$$a_j = \sum_i w_{ji} z_i \; and \; z_j = h(a_j)$$

- to find the activations of all the hidden and output units.
- Evaluate the δ_k for all the output units using

$$\delta_k = y_k - t_k$$

- Backpropagate the δ's using

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

- to obtain δ_j, for each hidden unit in the network.
- Use

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

- to evaluate the required derivatives.

Example

- Lets consider a two-layer network, together with a sum- of-squares error, in which the output units have linear activation functions, so that $y_k = a_k$, while the hidden units have logistic sigmoid activation function given by

$$h(a) \equiv tanh(a)$$

$$tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

- A useful feature of this function is that its derivative can be expressed in a particularly simple form:

$$h'(a) = 1 - h(a)^2$$

- We also consider a standard sum-of-squares error function, so that for pattern n the error is given by

$$E_n = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2$$

- where y_k is the activation of output unit k, and t_k is the corresponding target, for a particular input pattern x_n
- For each pattern in the training set in turn, we first perform a forward propagation using

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = \tanh(a_j)$$

$$y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

- Next we compute the δ's for each output unit using

$$\delta_k = y_k - t_k$$

- Then we backpropagate these to obtain δ's for the hidden units using

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k$$

- Finally, the derivatives with respect to the first-layer and second-layer weights are given by

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$