Machine Learning
DSECL ZG565

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Session 8
Date – 14th December 2019
Time – 9 am to 11 am

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Sugato Ghosal, Prof. S.K. H. Islam from BITS Pilani and many others who made their course materials freely available online.

Session 8: Review of session 1 to 7

- Session 1 Review: ML Introduction
- Session 2 Review: Math Prelims
- Session 3 Review: Bayesian Learning
- Session 4 Review: Naïve Bayes Classifier
- Session 5 Review: Logistic Regression
- Session 6 Review: Linear Regression
- Session 7 Review: Decision Tree and Random Forest

Session 1 Review: ML Introduction

What is Machine Learning?

Definition by Tom Mitchell (1998):

"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: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

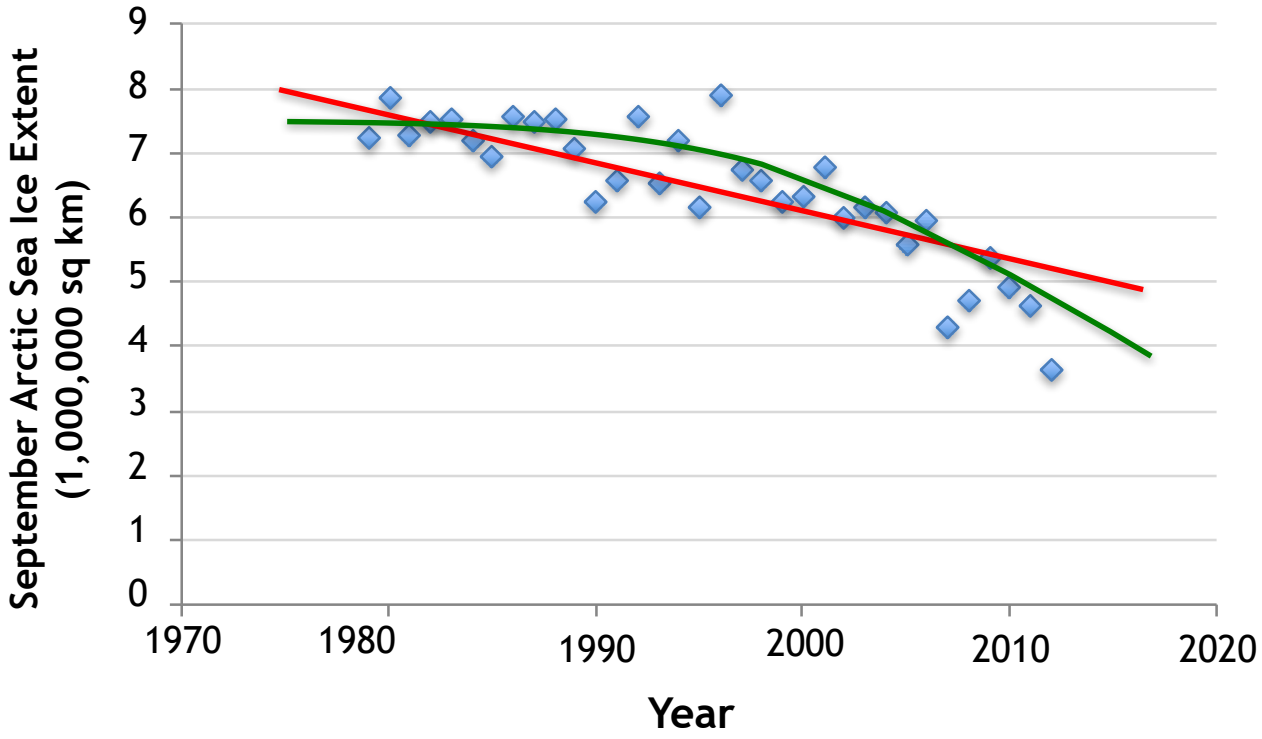
P = the probability that the program will win the next game.

A Checker Learning Problem

- **Task T:** Playing Checkers
- **Performance Measure P:** Percent of games won against opponents
- **Training Experience E:** To be selected ==> Games Played against itself

Supervised Learning: Regression

- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Learn a function $f(x)$ to predict y given x

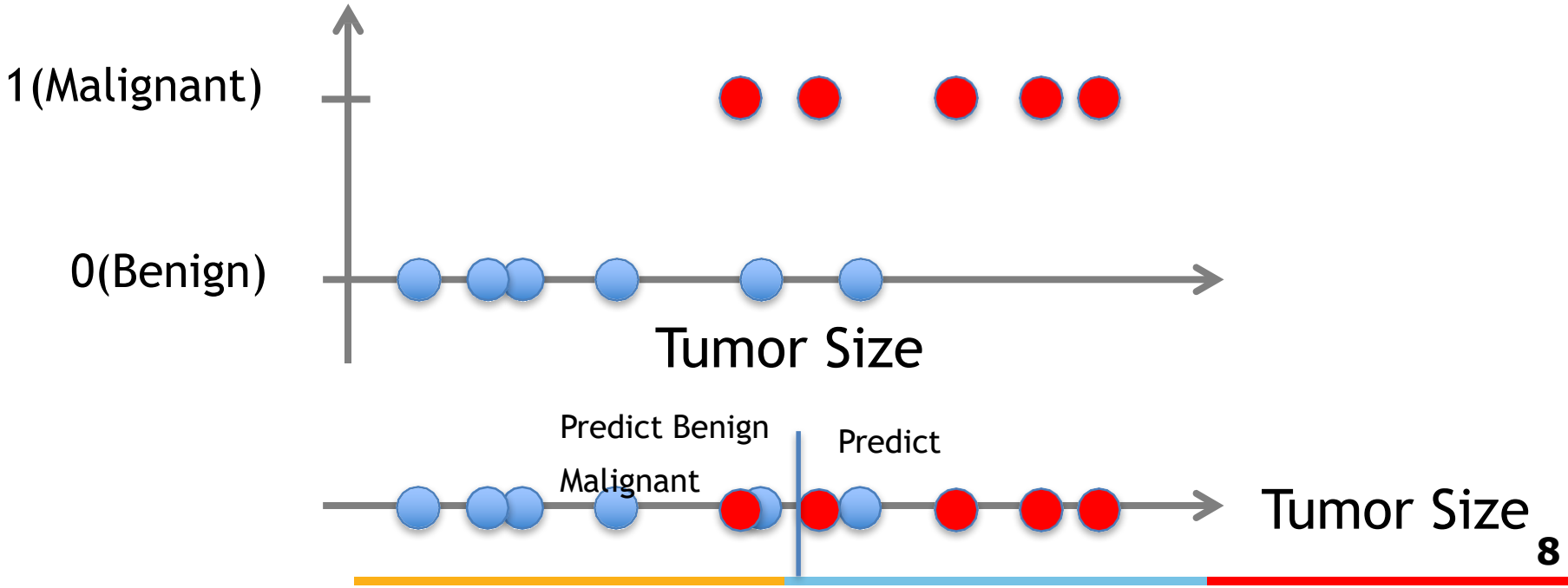


Data from G. Witt. Journal of Statistics Education, Volume 21, Number 1 (2013) Slide Credit: Eric Eaton

Supervised Learning: Classification

- Given (x_1, y_1), (x_2, y_2), …, (x_n, y_n)
- Learn a function $f(x)$ to predict y given x
  - y is categorical == classification

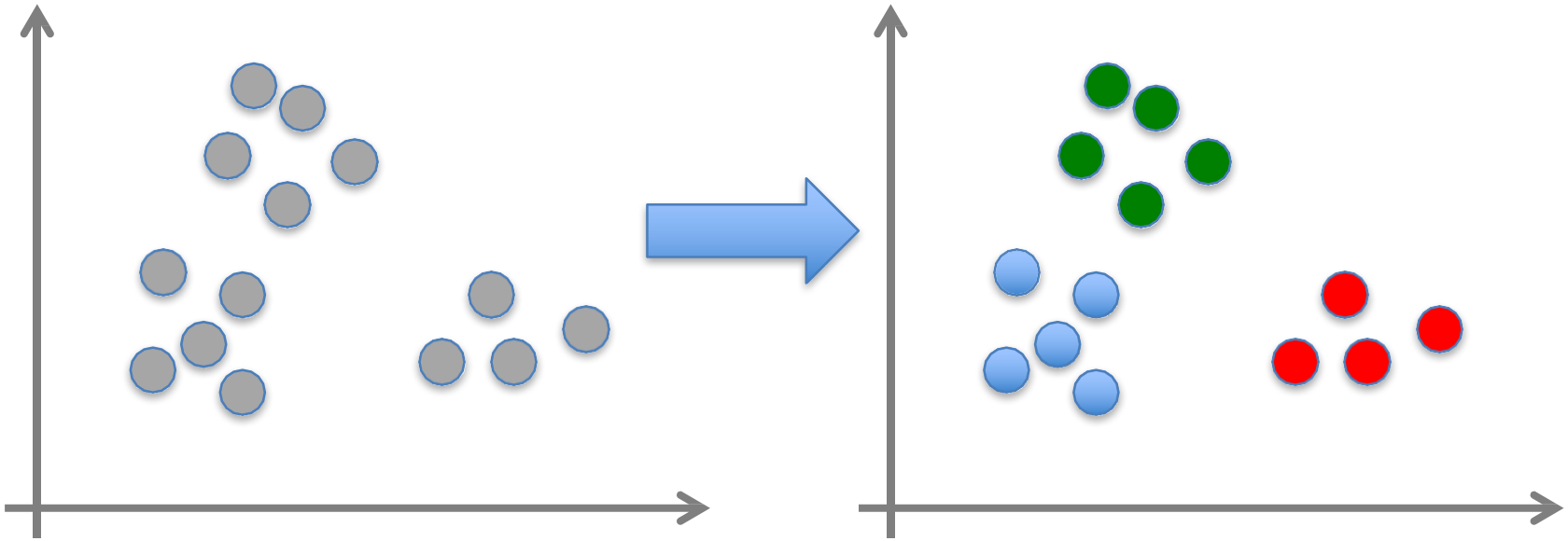
Breast Cancer (Malignant / Benign)



Slide Credit: Eric Eaton

Unsupervised Learning

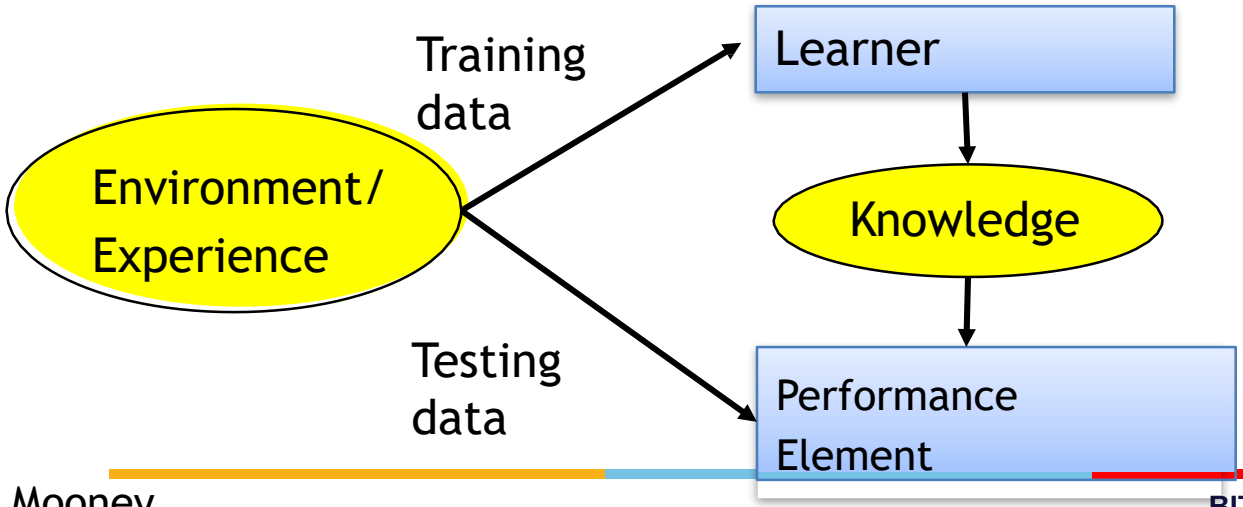
- Given $x_1, x_2, \ldots, x_n$ (without labels)
- Output hidden structure behind the x's
  - E.g., clustering



Slide Credit: Eric Eaton

Designing a Learning System

- Choose the training experience(data)
- Choose exactly what is to be learned
  - i.e. the ***target function***
- Choose how to represent the target function
- Choose a learning algorithm to infer the target function from the experience



Slide Credit: Ray Mooney

Issues in Machine Learning

- What algorithms are available for learning a concept? How well do they perform?
- How much training data is sufficient to learn a concept with high confidence?
- When is it useful to use prior knowledge?
- Are some training examples more useful than others?
- What are the best tasks for a system to learn?
- What is the best way for a system to represent its knowledge?

Session 2 Review: Math Prelims

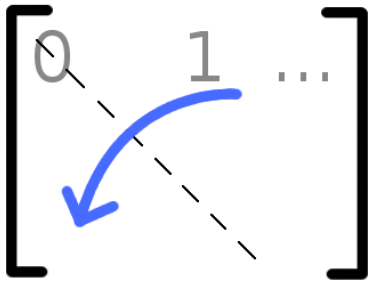
Inverse

- Given a matrix **A**, its inverse $\mathbf{A}^{-1}$ is a matrix such that $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
- E.g. $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$
- Inverse does not always exist. If $\mathbf{A}^{-1}$ exists, **A** is *invertible* or *non-singular*. Otherwise, it's *singular*.

BITS Pilani

Matrix Operations

- Transpose – flip matrix, so row 1 becomes column 1



$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

- A useful identity:

$$(ABC)^T = C^T B^T A^T$$

BITS Pilani

Differentiation Formulas

The following are common differentiation formulas:

- The derivative of a constant is 0.

$$\frac{d}{du}c = 0$$

- The derivative of a sum is the sum of the derivatives.

$$\frac{d}{du}(f(u) + g(u)) = f'(u) + g'(u)$$

More Formulas

- The derivative of *u* to a constant power:

$$\frac{d}{du} u^n = n * u^{n-1}$$

- The derivative of *e*:

$$\frac{d}{du} e^u = e^u$$

- The derivative of *log*:

$$\frac{d}{du} \log(u) = \frac{1}{u}$$

BITS Pilani

Product and Quotient

The product rule and quotient rules are commonly used in differentiation.

- Product rule:

$$\frac{d}{du}(f(u)*g(u)) = f(u)g'(u) + g(u)f'(u)$$

- Quotient rule:

$$\frac{d}{du}\left(\frac{f(u)}{g(u)}\right) = \frac{g(u)f'(u) - f(u)g'(u)}{g^{2}(u)}$$

Chain Rule

The chain rule allows you to combine any of the differentiation rules we have already covered.

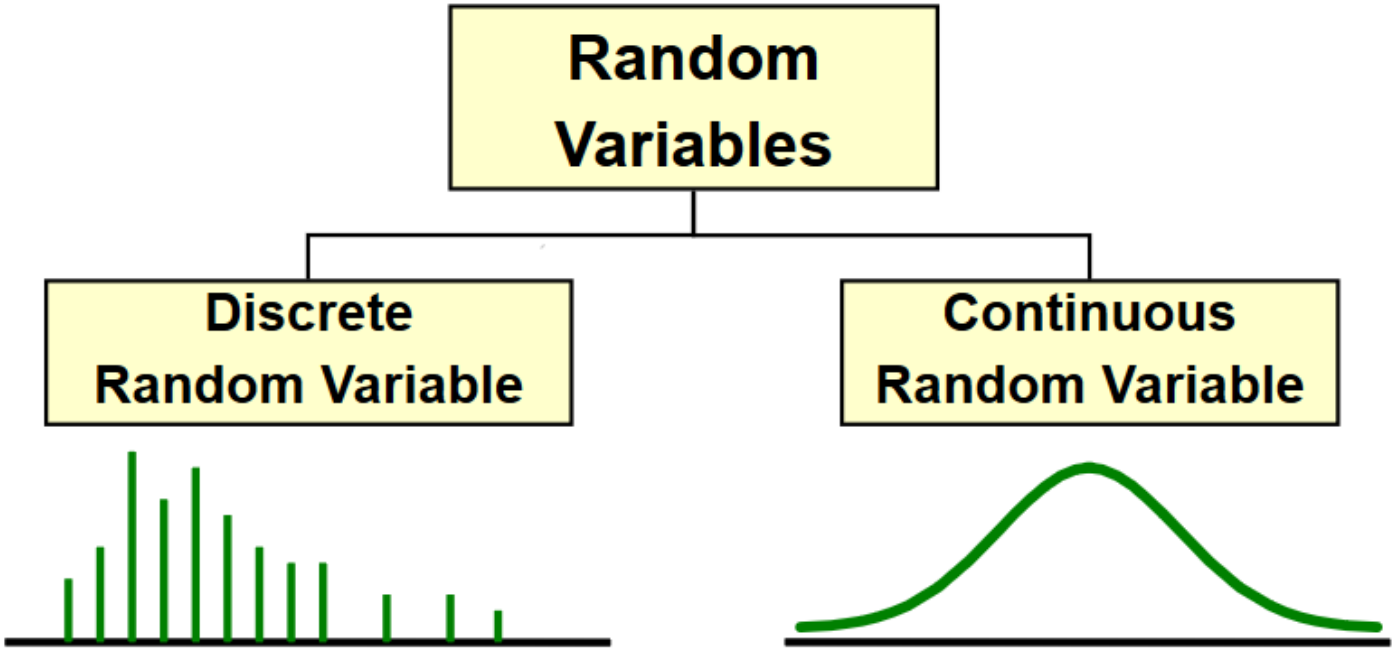
- First, do the derivative of the outside and then do the derivative of the inside.

$$\frac{d}{du} f(g(u)) = \quad f'(g(u))^* g'(u)^* du$$

BITS Pilani

Random Variable

Represents a possible numerical value from a random event



BITS Pilani

Random Variable

innovate achieve lead

Discrete Random Variable

- one that takes on a ***countable*** number of values
- usually count data [Number of]
- list **all** possible outcomes without missing any of them

Example:

- ✓ X = sum of values on the roll of two dice:
  X has to be either 2, 3, 4, …, or 12.

- ✓ Y = number of students in MTech DSE:
  Y has to be 60,65,70 ……

BITS Pilani

Random Variable

innovate achieve lead

Continuous Random Variable

- Variable that takes on an uncountable number of values
- Usually measurement data [time, weight, distance, etc]
- You can never list all possible outcomes even if you had an infinite amount of time

Example:

X = time it takes you to drive home from work place: X > 0, might be 30.1 minutes measured to the nearest tenth but in reality the actual time is 30.10000001……. minutes?)

Exercise: try to list all possible numbers between 0 and 1

BITS Pilani

Probability Theory

Notation

- A **random variable X** represents outcomes or states of the world.
- We will write p(x) to mean Probability(X = x)
- **Sample space**: the space of all possible outcomes (may be discrete, continuous, or mixed)
- p(x) is the **probability mass (density) function**
  - Assigns a number to each point in sample space
  - Non-negative, sums (integrates) to 1
  - Intuitively: how often does x occur, how much do we believe in x.

BITS Pilani

Probability Theory

Joint Probability Distribution

- Prob(X=x, Y=y)
  - "Probability of X=x and Y=y"
  - p(x, y)

Conditional Probability Distribution

- Prob(X=x|Y=y)
  - "Probability of X=x given Y=y"
  - p(x|y) = p(x,y)/p(y)

BITS Pilani

Probability Theory

if patient has meningitis, then very often a stiff neck is observed
$P(S|M) = 0.8$ (can be easily determined by counting)

observation: 'I have a stiff neck! Do I have meningitis?' (is it reasonable to be afraid?)
$P(M|S) = ?$

we need to now: $P(M) = 0.0001$ (one of 10000 people has meningitis) and $P(S) = 0.1$ (one out of 10 people has a stiff neck).

then:

$$P(M|S) = \frac{P(S|M)P(M)}{P(S)} = \frac{0.8 \times 0.0001}{0.1} = 0.0008$$

Keep cool. Not very likely

Decision Theory

- Suppose **x** is an input vector together with a corresponding vector **t** of target variables
- Goal: predict **t** given a new value for **x.**
- The joint probability distribution ***p(x, t)*** provides a complete summary of the uncertainty associated with these variables.
- Determination of ***p(x, t)*** from a set of training data is called ***inference*** and is a difficult problem.

BITS Pilani

Decision Theory

Inference step

Determine either $p(t|\mathbf{x})$ or $p(\mathbf{x}, t)$.

Decision step

For given $\mathbf{x}$, determine optimal t.

BITS Pilani

Inference and Decision

1st approach

- Determine the class-conditional densities ***p(x | C_k)*** for each class ***C_k*** individually.
- Separately infer the prior class probabilities ***p(C_k)*** . Then use Bayes' theorem

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}$$

$$p(\mathbf{x}) = \sum_k p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)$$

BITS Pilani

Inference and Decision

2nd approach

- Solve the inference problem of determining the posterior class probabilities $p(C_k|x)$, and then subsequently use decision theory to assign each new x to one of the classes.
- Approaches that model the posterior probabilities directly are called ***discriminative models***.

BITS Pilani

Inference and Decision

3rd approach

- Find a function ***f(x)***, called a ***discriminant function***, which maps each input ***x*** directly onto a class label.
- Example - For two-class problems, ***f(·)*** might be binary valued and such that ***f = 0*** represents class C_1 and ***f = 1*** represents class C_2. In this case, probabilities play no role.

BITS Pilani

Probability Densities

Continuous Probability Distribution

Let X be a continuous rv. Then a *probability distribution or probability density function (pdf)* of X is a function $f(x)$ such that for any two numbers a and b,

$$P(a \le X \le b) = \int_a^b f(x)dx$$

The graph of f is the *density curve*.

BITS Pilani

Probability Distributions

- The outcomes for random variables and their associated probabilities can be organized in to distributions
- Two types of distributions based on types of Random variables: Discrete and Continuous
- Discrete:
  - Binomial distributions
- Continuous
  - Gaussian, exponential, t, F, chi-squared distributions

Bernoulli Distribution

- A r.v. X is said to follow Bernoulli's distribution when there are only two possible outcomes
  - By convention either Success (1) or Failure (0)
  - And there is only one trial
  - Ex: tossing a coin at the start f the match
- Let p represents the probability of success and (1-p) represent the probability of failure, then the probability mass function is defined as

$$f(x) = \begin{cases} p & \text{if } x=1 \\ 1-p & x=0 \end{cases}$$

$$p^x (1-p)^{(1-x)}$$

Binomial Distribution

- Again binary outcomes, but for n independent trials
  - Probability of success (p) remains the same for all the trials
  - Probability of r success is given by

$P(X=r) = {}^{n}C_{r}\, p^{r}\, (1-p)^{n-r}$

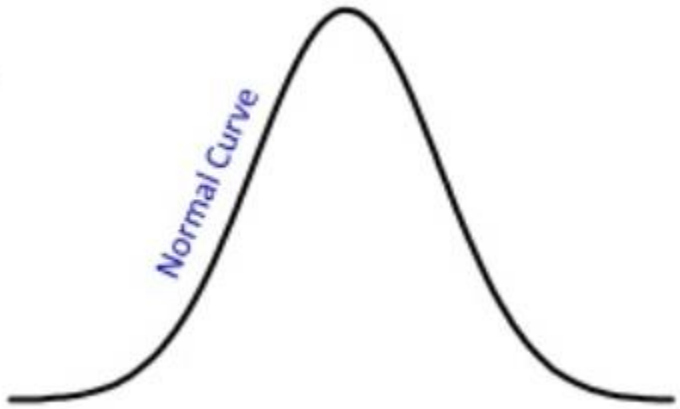
Mean $E(X) = np$

Variance $Var(X) = npq$ (where $q=1-p$)

Gaussian Distribution

Normal Curve:

- A graphical representation of the normal distribution.
- It is determined by the mean and the standard deviation.
- It a symmetric unimodal bell-shaped curve.
- Its tails extending infinitely in both directions.
- The wider the curve, the larger the standard deviation and the more variation exists in the process.
- The spread of the curve is equivalent to six times the standard deviation of the process.



BITS Pilani

Mean, Variance & Standard Deviation

- ✓ The mean of a discrete random variable is the ***weighted average*** of all of its values. The weights are the probabilities.

- ✓ This parameter is also called the expected value of X and is represented by E(X).

$$E(X) = \mu = \sum_{all\ x} xP(x)$$

- ✓ The variance is

$$V(X) = \sigma^2 = \sum_{all\ x} (x-\mu)^2 P(x)$$

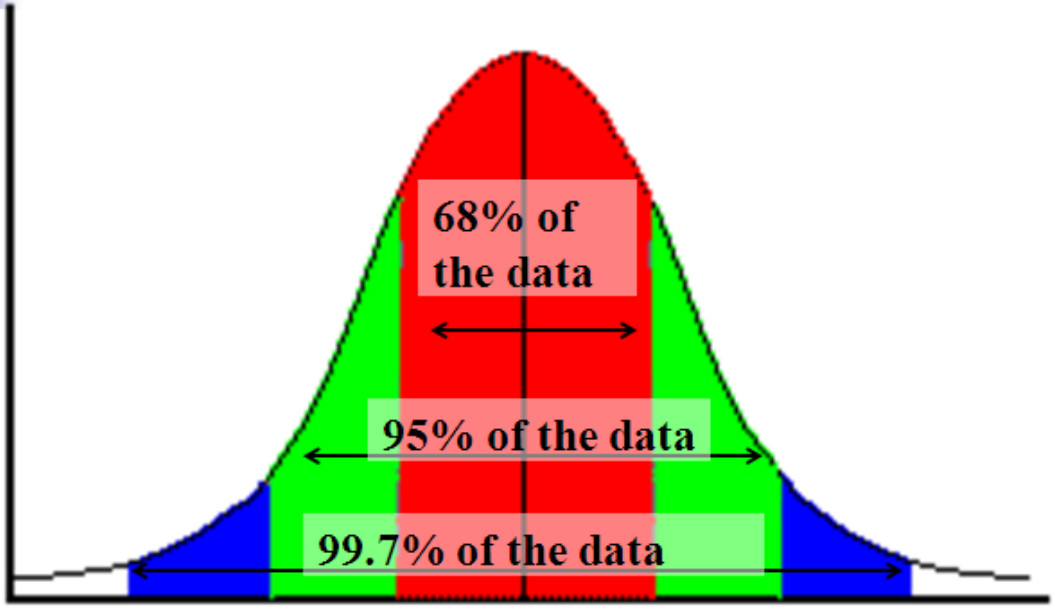
- ✓ The standard deviation is

$$\sigma = \sqrt{\sigma^2}$$

Gaussian Distribution

Empirical Rule:

- For any normally distributed data:
  - **68%** of the data fall within **1** standard deviation of the mean.
  - **95%** of the data fall within **2** standard deviations of the mean.
  - **99.7%** of the data fall within **3** standard deviations of the mean.

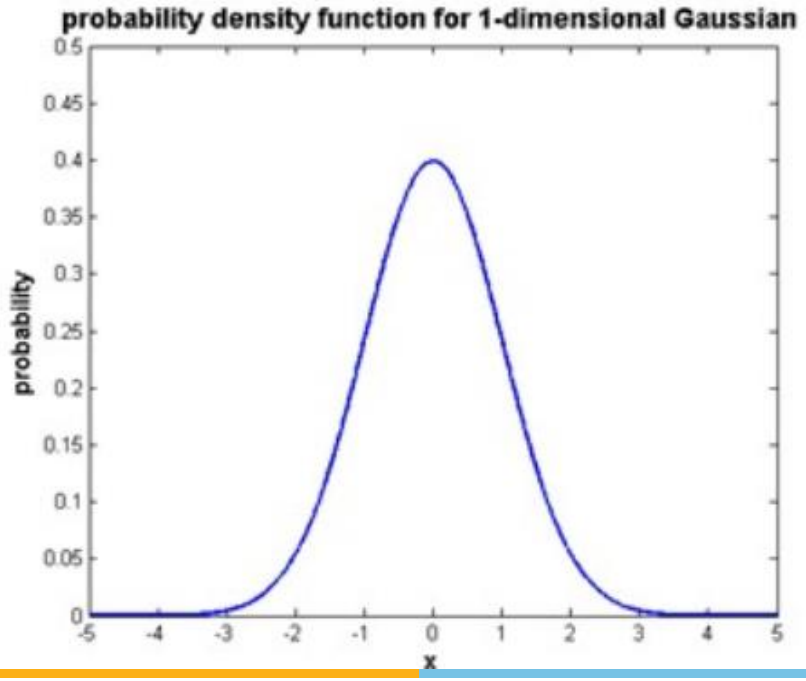


BITS Pilani

Gaussian Distribution

In one dimension

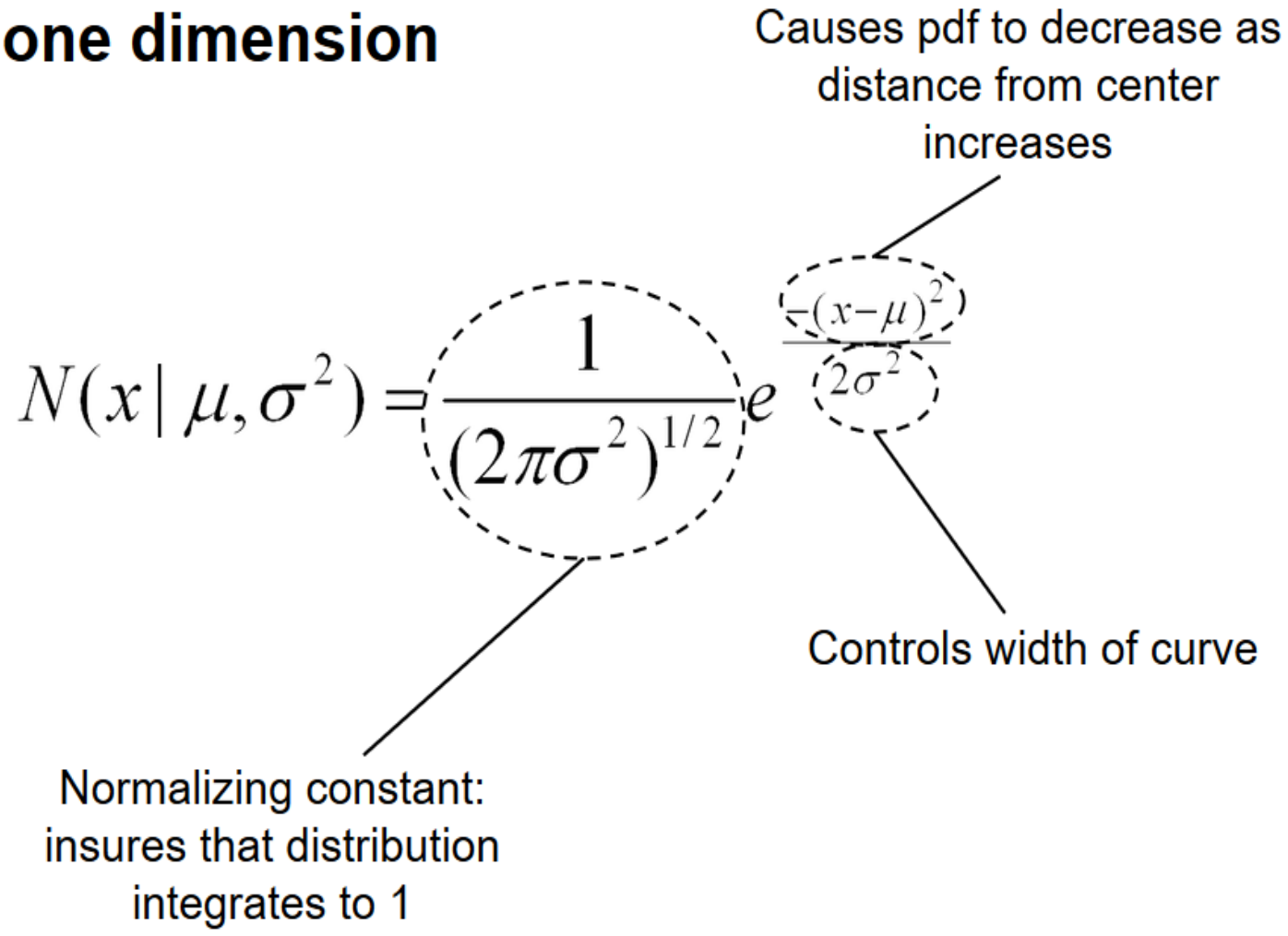
$$N(x \mid \mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



BITS Pilani

Gaussian Distribution

In one dimension



BITS Pilani

JOINT Distributions

- Probability distribution of two random variables X $\{x_1, x_2, \ldots x_n\}$ and Y$\{y_1, y_2 .. y_k\}$
  - Occurrence of X=xi and Y=yi together
- Example:
  - P(X=0, Y<=1)
  - P(X=1)

$$= \sum_{y=0}^{2} P(X = 1, Y)$$

$$= 1/6 + 1/6 + 1/8$$

X \ Y	0	1	2
0	1/4	1/6	1/8
1	1/6	1/6	1/8

JOINT Distribution

- Marginal Distribution
  - Sum over any one variable is called Marginal Distribution

$$P(X=x) = \sum_{y \in Y} P(X, Y)$$

$$P(Y=y) = \sum_{x \in X} P(X, Y)$$

Conditional Probability

- Estimating probability for two or more related events
  - Measure influence of one variable over another
- Let A and B be two events, $p(B) > 0$

  $$p(A|B) = p(A \cap B) / p(B)$$
- Using random variable notations,
  - p(a|b) denotes the probability of A=a and B=b
    - i.e. p(A=a |B=b)

Basic Formulas for Probabilities

- *Product Rule*: probability $P(A \wedge B)$ of a conjunction of two events A and B:

$$P(A \wedge B) = P(A \mid B)\, P(B) = P(B \mid A)\, P(A)$$

- *Sum Rule*: probability of a disjunction of two events A and B:

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

- *Theorem of total probability*: if events $A_1, \ldots, A_n$ are mutually exclusive with $\Sigma_{i=1}^{n} P(A_i) = 1$, then

$$P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$$

Example

- Does a patient have cancer or not?
  - The patient takes a lab test and the test returns a correct positive result in only 98% of the cases in which the disease is actually present
  - And a correct negative result in only 97% of the cases in which the disease is not present
  - Of the total population, only 0.008% has cancer

Example

Given:

$P(cancer) =$ $P(\neg cancer) =$

$P(+|cancer) =$ $P(-|cancer) =$

$P(+|\neg cancer) =$ $P(-|\neg cancer) =$

Using Bayes Theorem:

$P(\text{cancer}|+) = P(+|cancer)\ P(cancer) / P(+)$

$P(\neg cancer\ |+) = P(+|\ \neg cancer)\ P(\neg cancer) / P(+)$

Since denominator is common

$P(\text{cancer}|+)$ *proportional to* $P(+|cancer)\ P(cancer)$

$P(\neg cancer\ |+)$ *proportional to* $P(+|\ \neg cancer)\ P(\neg cancer)$

Example

$P(cancer) = 0.008$ $P(\neg cancer) = 1\text{-}0.008=0.992$

$P(+|cancer) = 0.98$ $P(-|cancer) = 0.02$

$P(+|\neg cancer) = 1\text{-}0.97=0.3$ $P(-|\neg cancer) = 0.97$

Remember: Some terminology

- Likelihood function: P(data | y)
- Prior: P(y)
- Posterior: P(y | data)

- **Conjugate prior**: P(y) is the conjugate prior for likelihood function P(data | y) if the forms of P(y) and P(y | data) are the same.

Bayesian Learning

- Bayesian learning algorithms that calculate explicit probabilities for hypotheses, such as the naive Bayes classifier, are among the most practical approaches to certain types of learning problems

- For example: Problem of learning to classify text documents such as electronic news articles.
- For such learning tasks, the naive Bayes classifier is among the most effective algorithms known

Features of Bayesian learning

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct.
- Flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions (e.g., hypotheses such as "this pneumonia patient has a 93% chance of complete recovery").

Features of Bayesian learning

- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis.
- Prior knowledge is provided by asserting
  - prior probability for each candidate hypothesis, and
  - probability distribution over observed data for each possible hypothesis.
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.

Hypothesis

- Relationship between the input and output values.
- Lets say that **target function** $y=f(\mathbf{x})$ However, f(.) is unknown function to us.
- Machine learning algorithms try to guess a hypothesis function $h(\mathbf{x})$ that approximates the unknown $f(.)$
- Set of all possible hypotheses is known as the Hypothesis set or space H(.)
- Goal is the learning process is to find the final hypothesis that best approximates the unknown target function.

Bayes Theorem

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

- $P(h)$ = prior probability of hypothesis h
- $P(D)$ = prior probability of training data D
- $P(h \mid D)$ = probability of h given D
- $P(D \mid h)$ = probability of D given h

MAP Hypothesis

- Using Bayes theorem, we compute the MAP hypothesis for all probable hypothesis (or all unique class labels)
- Identify the best hypothesis describing the data as

$$h_{MAP} = argmax_{h \in H} P(h|D)$$

$$= argmax_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= argmax_{h \in H} P(D|h)P(h)$$

H: set of all hypothesis

P(D) is independent of h and is same for all hypothesis, therefore dropped

Maximum Likelihood estimation

- When no prior information is available, all hypothesis are equally likely i.e. p(hi) = p(hj)
  - This is also true for a balanced class problem where all the classes are equally likely
  - This is known as Uniform prior
  - MAP hypothesis further simplifies to:

$$H_{ML} = argmax_{h \in H} \mathrm{P}(\mathrm{D}|\mathrm{h})$$

This is called Maximum Likelihood Hypothesis

Minimum Description Length Principle

$$
\begin{aligned}
h_{MAP} &= \arg\max_{h \in H} P(D|h)P(h) \\
&= \arg\max_{h \in H} \log_2 P(D|h) + \log_2 P(h) \\
&= \arg\min_{h \in H} -\log_2 P(D|h) - \log_2 P(h) \quad (1)
\end{aligned}
$$

Interesting fact from information theory:

The optimal (shortest expected coding length) code for an event with probability p is $-\log_2 p$ bits.

So interpret (1):

- $L_{C1}(h) = length(h) = -\log_2 P(h)$
- $L_{C2}(D|h) = length(misclassifications) = -\log_2 P(D|h)$

$\rightarrow$ prefer the hypothesis that minimizes

$$length(h) + length(misclassifications)$$

Most Probable Classification of New Instances

- So far we've sought the most probable *hypothesis* given the data D (i.e., h_{MAP})
- Given new instance x, what is its most probable *classification*?
  - $h_{MAP}(x)$ is not the most probable classification!
- Consider:
  - Three possible hypotheses:

$$P(h_1|D) = .4,\ P(h_2|D) = .3,\ \ P(h_3|D) = .3$$

  - Given new instance x,

$$h_1(x) = +,\ \ h_2(x) = -,\ \ h_3(x) = -$$

  - What's most probable classification of x?

Bayes Optimal Classifier

- **Bayes optimal classification:**

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

- Example:

$P(h_1|D) = .4,\ P(-|h_1) = 0,\ P(+|h_1) = 1$
$P(h_2|D) = .3,\ P(-|h_2) = 1,\ P(+|h_2) = 0$
$P(h_3|D) = .3,\ P(-|h_3) = 1,\ P(+|h_3) = 0$

therefore

$$\sum_{h_i \in H} P(+|h_i)P(h_i|D) = .4$$
$$\sum_{h_i \in H} P(-|h_i)P(h_i|D) = .6$$

and

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) = -$$

Gibbs Classifier

- Bayes optimal classifier provides best result, but can be expensive if many hypotheses.
- Gibbs algorithm:
  1. Choose one hypothesis at random, according to $P(h|D)$
  2. Use this to classify new instance
- Surprising fact: Assume target concepts are drawn at random from H according to priors on H. Then:

$$E[error_{Gibbs}] \leq 2E[error_{BayesOptional}]$$

- Suppose correct, uniform prior distribution over H, then
  - Pick any hypothesis from VS, with uniform probability
  - Its expected error no worse than twice Bayes optimal

Practical Issues of Bayesian learning

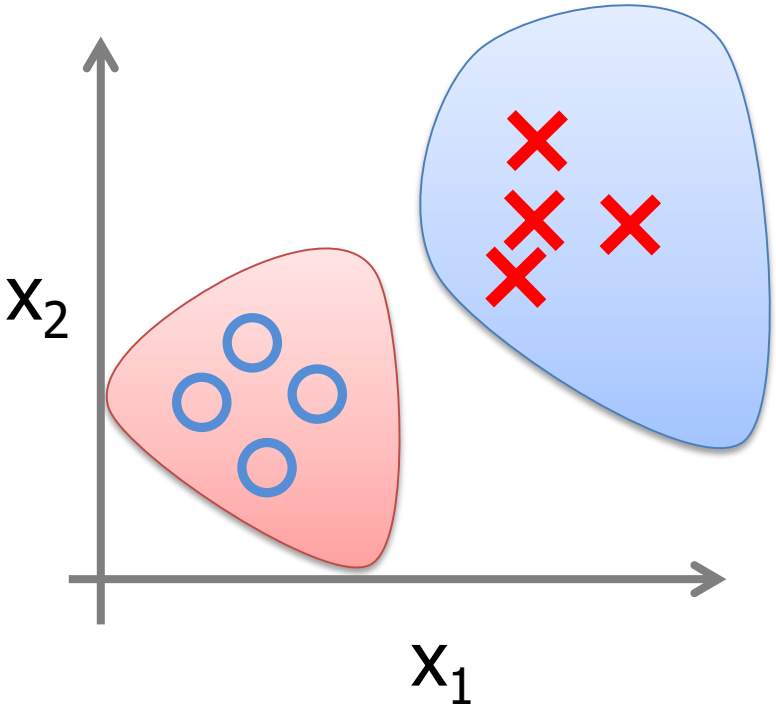
- Require initial knowledge of many probabilities
  - Often estimated based on background knowledge, previously available data, and assumptions about the form of the underlying distributions.
- Significant computational cost required to determine the Bayes optimal hypothesis in the general case (linear in the number of candidate hypotheses)

Probabilistic Generative Classifiers

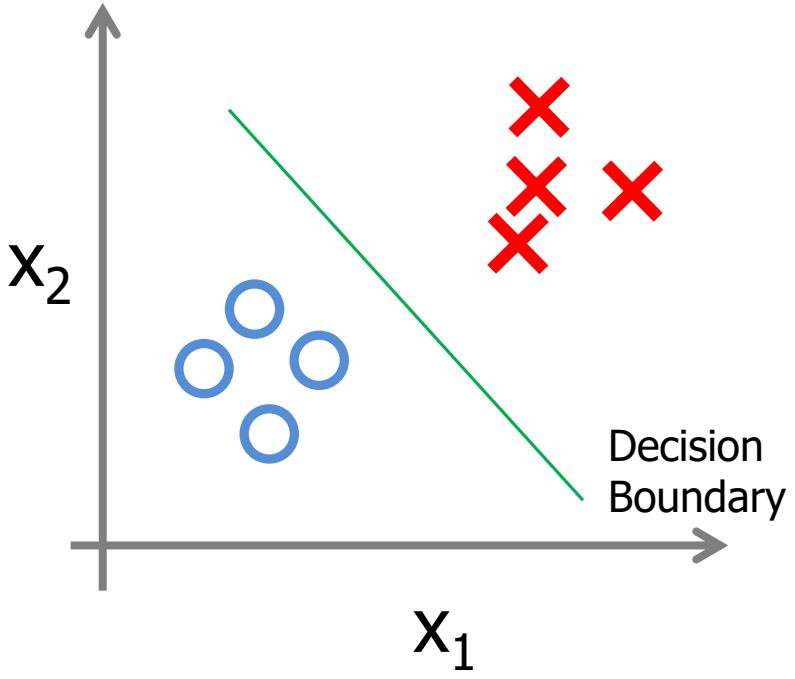
- Approach is to understand the process that generated the data
  - Generative Models P(X,Y)
  - Build a model for all the positive cases or category 1
  - Build another model for all the negative cases or category 2
  - For predicting a new test case
    - Run the test case with both the models and choose the model with maximum probability

Probabilistic Generative Model versus Probabilistic Discriminative Model

Generative:



Discriminative:



Probabilistic Models: Generative/Discriminative

- Model $p(C_k|\boldsymbol{x})$ in an *inference* stage and use it to make optimal decisions
- Approaches to computing the $p(C_k|\boldsymbol{x})$
  1. Generative
     - Model class conditional densities by $p(\boldsymbol{x}|C_k)$ together with prior probabilities $p(C_k)$
     - Then use Bayes rule to compute posterior

$$p(C_k \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid C_k)p(C_k)}{p(\boldsymbol{x})}$$

  2. Discriminative
     - Directly model conditional probabilities $p(C_k|\boldsymbol{x})$

13

Generative Models

- Generative models
  - Build model to estimate the posterior probability P(Y|X) by estimating
  - likelihood of data given target (hypothesis) P(X|Y)
  - Prior probabilities over target P(Y)
  - In general, for a specific class Y=ck,

$$P(Y = c_k|X) = \frac{P(X|Y = c_k) * P(Y=c_k)}{P(X)}$$

Conditional independence

- **Definition**: X is conditionally independent of Y given Z, if the probability distribution governing X is independent of the value of Y, given the value of Z

$$(\forall i, j, k)\ P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z_k)$$

$$P(X|Y, Z) = P(X|Z)$$

Example:

$$P(\text{Thunder}|\text{Rain}, \text{Lightning}) = P(\text{Thunder}|\text{Lightning})$$

Slide credit: Tom Mitchell

Applying conditional independence

- Naïve Bayes assumes X_i are conditionally independent given Y

e.g., $P(X_1|X_2, Y) = P(X_1|Y)$

$$P(X_1, X_2|Y) = P(X_1|X_2, Y)P(X_2|Y)$$
$$= P(X_1|Y)P(X_2|Y)$$

General form: $P(X_1, \cdots, X_n|Y) = \prod_{j=1}^{n} P(X_j|Y)$

Slide credit: Tom Mitchell

Naïve Bayes Independence assumption

- Assumption:

$$P(X_1, \cdots, X_n|Y) = \prod_{j=1}^{n} P(X_j|Y)$$

- i.e., X_i and X_j are conditionally independent given Y for $i \neq j$

Slide credit: Tom Mitchell

Naïve Bayes classifier

- Bayes rule:

$$P(Y = y_k | X_1, \cdots, X_n) = \frac{P(Y = y_k) P(X_1, \cdots, X_n | Y = y_k)}{\sum_j P(Y = y_j) P(X_1, \cdots, X_n | Y = y_j)}$$

- Assume conditional independence among X_i's:

$$P(Y = y_k | X_1, \cdots, X_n) = \frac{P(Y = y_k) \Pi_i P(X_i | Y = y_k)}{\sum_j P(Y = y_j) \Pi_i P(X_i | Y = y_j)}$$

- Pick the most probable (MAP) Y

$$\hat{Y} \leftarrow \underset{y_k}{\operatorname{argmax}} P(Y = y_k) \Pi_i P(X_i | Y = y_k)$$

Prior Probability

MLE

Slide credit: Tom Mitchell

NAÏVE BAYES CLASSIFIER

- Assume independence among attributes X_i when class is given:
  - $P(X_1, X_2, \ldots, X_d \mid Y_j) = P(X_1 \mid Y_j)\, P(X_2 \mid Y_j) \ldots P(X_d \mid Y_j)$
  - Now we can estimate $P(X_i \mid Y_j)$ for all X_i and Y_j combinations from the training data
  - New point is classified to Y_j if $P(Y_j) \prod P(X_i \mid Y_j)$ is maximal.

Slide adopted from "Introduction to Data mining" Vipin Kumar

Naive Bayes Classifier

- Assume target function $f : X \rightarrow V$, where each instance x described by attributes $<a_1, a_2 \ldots a_n>$.
- Most probable value of $f(x)$ is:

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n)$$

$$v_{MAP} = \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j) P(v_j)}{P(a_1, a_2 \ldots a_n)}$$

$$= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j) P(v_j)$$

Naive Bayes assumption:

$$P(a_1, a_2 \ldots a_n | v_j) = \prod_i P(a_i | v_j)$$

Naive Bayes classifier:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_i P(a_i | v_j)$$

Naive Bayes Algorithm

- Naive Bayes Learn(*examples*)

  For each target value v_j

  $$\hat{P}(v_j) \leftarrow \text{estimate } P(v_j)$$

  For each attribute value a_i of each attribute a

  $$\hat{P}(a_i \mid v_j) \leftarrow \text{estimate } P(a_i \mid v_j)$$

- Classify New Instance(*x*)

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i | v_j)$$

Example 1

Name	Give Birth	Can Fly	Live in Water	Have Legs	Class
human	yes	no	no	yes	mammals
python	no	no	no	no	non-mammals
salmon	no	no	yes	no	non-mammals
whale	yes	no	yes	no	mammals
frog	no	no	sometimes	yes	non-mammals
komodo	no	no	no	yes	non-mammals
bat	yes	yes	no	yes	mammals
pigeon	no	yes	no	yes	non-mammals
cat	yes	no	no	yes	mammals
leopard shark	yes	no	yes	no	non-mammals
turtle	no	no	sometimes	yes	non-mammals
penguin	no	no	sometimes	yes	non-mammals
porcupine	yes	no	no	yes	mammals
eel	no	no	yes	no	non-mammals
salamander	no	no	sometimes	yes	non-mammals
gila monster	no	no	no	yes	non-mammals
platypus	no	no	no	yes	mammals
owl	no	yes	no	yes	non-mammals
dolphin	yes	no	yes	no	mammals
eagle	no	yes	no	yes	non-mammals

Give Birth	Can Fly	Live in Water	Have Legs	Class
yes	no	yes	no	?

A: attributes

M: mammals

N: non-mammals

$$P(A|M) = \frac{6}{7} \times \frac{6}{7} \times \frac{2}{7} \times \frac{2}{7} = 0.06$$

$$P(A|N) = \frac{1}{13} \times \frac{10}{13} \times \frac{3}{13} \times \frac{4}{13} = 0.0042$$

$$P(A|M)P(M) = 0.06 \times \frac{7}{20} = 0.021$$

$$P(A|N)P(N) = 0.004 \times \frac{13}{20} = 0.0027$$

P(A|M)P(M) > P(A|N)P(N)

=> Mammals

Issues with Naïve Bayes Classifier

Consider the table with Tid = 7 deleted

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

Naïve Bayes Classifier:

P(Refund = Yes | No) = 2/6
P(Refund = No | No) = 4/6
P(Refund = Yes | Yes) = 0
P(Refund = No | Yes) = 1
P(Marital Status = Single | No) = 2/6
P(Marital Status = Divorced | No) = 0
P(Marital Status = Married | No) = 4/6
P(Marital Status = Single | Yes) = 2/3
P(Marital Status = Divorced | Yes) = 1/3
P(Marital Status = Married | Yes) = 0/3
For Taxable Income:
If class = No: sample mean = 91
sample variance = 685
If class = No: sample mean = 90
sample variance = 25

Given X = (Refund = Yes, Divorced, 120K)

$P(X \mid No) = 2/6 \times 0 \times 0.0083 = 0$

$P(X \mid Yes) = 0 \times 1/3 \times 1.2 \times 10^{-9} = 0$

Naïve Bayes will not be able to classify X as Yes or No!

Issues with Naïve Bayes Classifier

- If one of the conditional probabilities is zero, then the entire expression becomes zero
- Need to use other estimates of conditional probabilities than simple fractions
- Probability estimation:

$$\text{Original}: P(A_i \mid C) = \frac{N_{ic}}{N_c}$$

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

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

c: number of classes

p: prior probability of the class

m: parameter

N_c: number of instances in the class

N_{ic}: number of instances having attribute value A_i in class c

A Simple Example

Text	Tag
"A great game"	Sports
"The election was over"	Not sports
"Very clean match"	Sports
"A clean but forgettable game"	Sports
"It was a close election"	Not sports

Which tag does the sentence *A very close game* belong to? i.e. P(sports| *A very close game*)

Feature Engineering: Bag of words i.e use word frequencies without considering order

Using Bayes Theorem:

P(sports| *A very close game*)

$$= \frac{\textbf{P(}\textit{A very close game}\textbf{| sports) P(sports)}}{\textbf{P(}\textit{A very close game}\textbf{)}}$$

We assume that every word in a sentence is **independent** of the other ones

$$P(a\,very\,close\,game) = P(a) \times P(very) \times P(close) \times P(game)$$

$$P(a\,very\,close\,game|Sports) = P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports)$$

"close" doesn't appear in sentences of sports tag, So P(close | sports) = 0, which makes product 0

Laplace smoothing

- Laplace smoothing: we add 1 or in general constant k to every count so it's never zero.
- To balance this, we add the number of possible words to the divisor, so the division will never be greater than 1
- In our case, the possible words are ['a', 'great', 'very', 'over', 'it', 'but', 'game', 'election', 'clean', 'close', 'the', 'was', 'forgettable', 'match'].

Apply Laplace Smoothing

Word	P(word | Sports)	P(word | Not Sports)
a	2+1 / 11+14	1+1 / 9+14
very	1+1 / 11+14	0+1 / 9+14
close	0+1 / 11+14	1+1 / 9+14
game	2+1 / 11+14	0+1 / 9+14

$$P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports) \times P(Sports)$$
$$= 2.76 \times 10^{-5}$$
$$= 0.0000276$$

$$P(a|Not\,Sports) \times P(very|Not\,Sports) \times P(close|Not\,Sports) \times P(game|Not\,Sports) \times P(Not\,Sports)$$
$$= 0.572 \times 10^{-5}$$
$$= 0.00000572$$

Naive Bayes Classifier

- Along with decision trees, neural networks, one of the most practical learning methods.
- When to use
  - Moderate or large training set available
  - Attributes that describe instances are conditionally independent given classification
- Successful applications:
  - Diagnosis
  - Classifying text documents

Learning to Classify Text

- Why?
  - Learn which news articles are of interest
  - Learn to classify web pages by topic

- Naive Bayes is among most effective algorithms
- What attributes shall we use to represent text documents??

Learning to Classify Text

LEARN_NAIVE_BAYES_TEXT (*Examples*, V)

1. *collect all words and other tokens that occur in Examples*

- *Vocabulary* ← all distinct words and other tokens in *Examples*

2. *calculate the required* $P(v_j)$ *and* $P(w_k \mid v_j)$ *probability terms*

- For each target value v_j in V do
  - $docs_j \leftarrow$ subset of *Examples* for which the target value is v_j
  - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
  - $Text_j \leftarrow$ a single document created by concatenating all members of $docs_j$

Learning to Classify Text

- $n \leftarrow$ total number of words in $Text_j$ (counting duplicate words multiple times)
- for each word w_k in *Vocabulary*
  - $n_k \leftarrow$ number of times word w_k occurs in $Text_j$
  - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

CLASSIFY_NAIVE_BAYES_TEXT (*Doc*)

- *positions* $\leftarrow$ all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return v_{NB} where $v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$

Naïve Bayes for Text Classification

- **Naïve Bayes assumption helps a lot!**
  - $P(X_i = x_i | Y = y)$ is just the probability of observing word x_i at the ith position in a document on topic y.
  - Assume X_i is independent of all other words in document given the label y: $P(X_i = x_i | Y = y, X_{-i}) = P(X_i = x_i | Y = y)$.

$$h_{NB}(x) = \arg\max_y P(y) \prod_{i=1}^{lengthDoc} P(X_i = x_i | y)$$

  - For each label y, have 1000 distributions of size 10000 to estimate.
  - This is 10000×1000 items, which is big but much less than 10000^{1000}...

Logistic Regression

Idea:

- Naïve Bayes allows computing P(Y|X) by learning P(Y) and P(X|Y)

- Why not learn P(Y|X) directly?

Sigmoid/Logistic Function

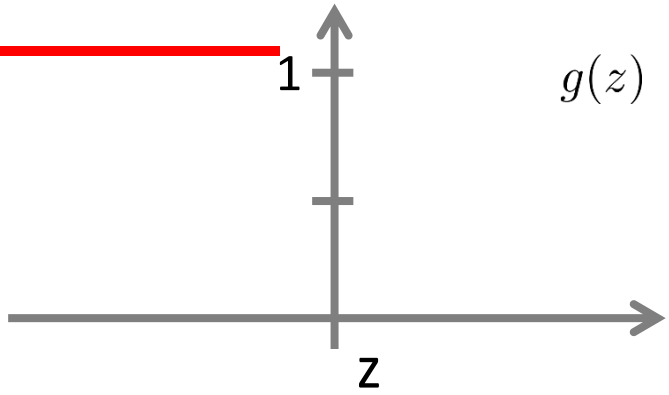
- Sigmoid/logistic function takes a real value as input and outputs another value between 0 and 1
- That framework is called logistic regression
  - Logistic: A special mathematical sigmoid function it uses
  - Regression: Combines a weight vector with observations to create an answer

$$h_\theta(x) = g(\theta^T x)$$

Logistic Regression

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$



Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

predict "$y = 0$" if $h_\theta(x) < 0.5$

Learning model parameters

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix}$ $x_0 = 1, y \in \{0, 1\}$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters (feature weights) θ ?

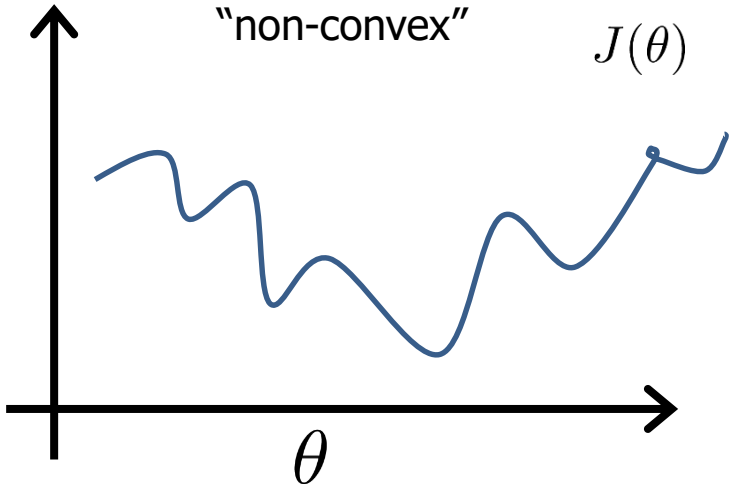
Error (Cost) Function

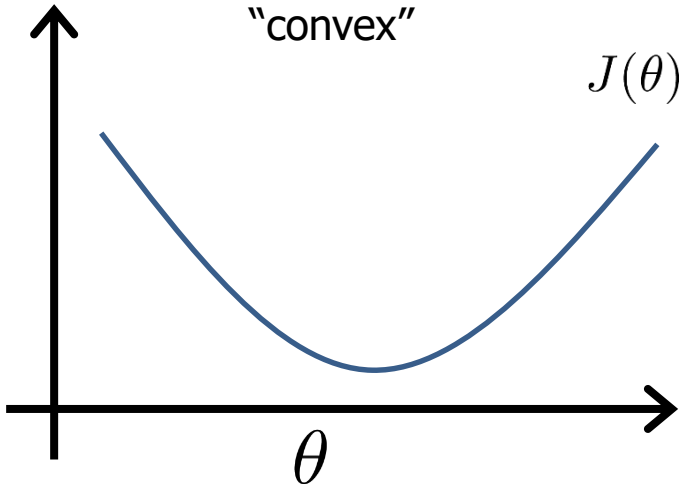
- Our prediction function is non-linear (due to sigmoid transform)
- Squaring this prediction as we do in MSE results in a non-convex function with many local minima.
- If our cost function has many local minimums, gradient descent may not find the optimal global minimum.
- So instead of Mean Squared Error, we use a error/ cost function called Cross-Entropy, also known as Log Loss.

MSE Cost Function

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

$$\text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

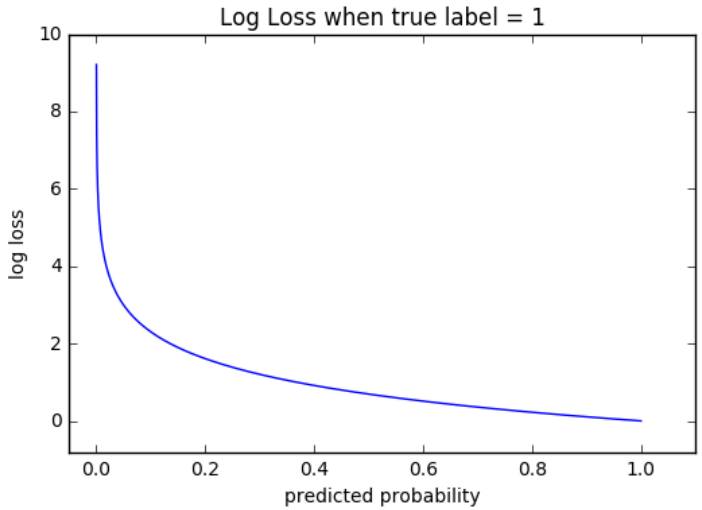




Cross Entropy

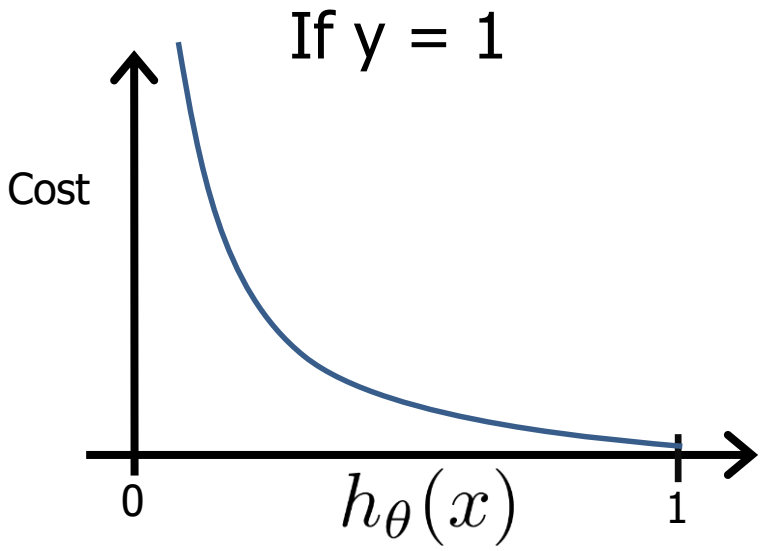
- Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.
- Cross-entropy loss increases as the predicted probability diverges from the actual label.
- So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value.
- A perfect model would have a log loss of 0.

- Cross-entropy loss can be divided into two separate cost functions:
  one for y=1 and
  one for y=0.



Logistic regression cost function (cross entropy)

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

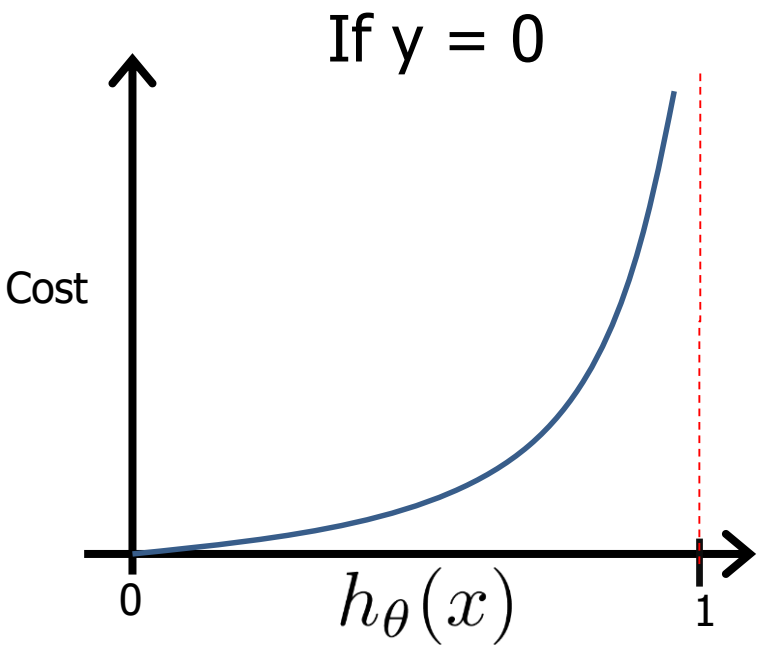


$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Logistic regression cost function

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



Cost=0; If y=0 and $h_\theta(x)=0$

Cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

To fit parameters θ : Apply Gradient Descent Algorithm

$$\min_\theta J(\theta)$$

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Logistic Regression

- Consider learning f: X → Y, where
  - X is a vector of real-valued features, $< X_1 \ldots X_n >$
  - Y is boolean
  - assume all X_i are conditionally independent given Y
  - model $P(X_i \mid Y = y_k)$ as Gaussian $N(\mu_{ik},\sigma_i)$
  - model P(Y) as Bernoulli (two-point) distribution(π)

- What does that imply about the form of P(Y|X)?

$$P(Y = 1|X =< X_1, \ldots X_n >) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

Very convenient!

$$P(Y = 1|X =< X_1, ...X_n >) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

implies

$$P(Y = 0|X =< X_1, ...X_n >) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

implies

$$\frac{P(Y = 0|X)}{P(Y = 1|X)} = exp(w_0 + \sum_i w_i X_i)$$

implies

$$\ln \frac{P(Y = 0|X)}{P(Y = 1|X)} = w_0 + \sum_i w_i X_i$$

Very convenient!

$$P(Y=1|X=<X_1,...X_n>)=\frac{1}{1+exp(w_0+\sum_i w_iX_i)}$$

implies

$$P(Y=0|X=<X_1,...X_n>)=\frac{exp(w_0+\sum_i w_iX_i)}{1+exp(w_0+\sum_i w_iX_i)}$$

implies

$$\frac{P(Y=0|X)}{P(Y=1|X)}=exp(w_0+\sum_i w_iX_i)$$

linear classification rule!

implies

$$\ln\frac{P(Y=0|X)}{P(Y=1|X)}=w_0+\sum_i w_iX_i$$

Logistic Regression and Gaussian Naïve Bayes Classifier

- Interestingly, the parametric form of P(Y|X) used by Logistic Regression is precisely the form implied by the assumptions of a Gaussian Naive Bayes classifier.

- Therefore, we can view Logistic Regression as a closely related alternative to GNB, though the two can produce different results in many cases

Estimating parameters

- we have L training examples: $\{\langle X^1, Y^1\rangle, \ldots \langle X^L, Y^L\rangle\}$
- maximum likelihood estimate for parameters W

$$= \arg\max_W \prod_l P(< X^l, Y^l > |W)$$

- maximum conditional likelihood estimate

Training Logistic Regression: MCLE

- Choose parameters W=<w_0, ... w_n> to <u>maximize conditional likelihood</u> of training data

where

$$P(Y = 0|X, W) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

- Training data D = $\{\langle X^1, Y^1\rangle, \ldots \langle X^L, Y^L\rangle\}$
- Data likelihood = $\prod_l P(X^l, Y^l|W)$
- Data <u>conditional</u> likelihood = $\prod_l P(Y^l|X^l, W)$

$$W_{MCLE} = \arg\max_W \prod_l P(Y^l|W, X^l)$$

Expressing Conditional Log Likelihood

$$l(W) \equiv \ln \prod_l P(Y^l|X^l, W) = \sum_l \ln P(Y^l|X^l, W)$$

$$P(Y = 0|X, W) = \frac{1}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{exp(w_0 + \sum_i w_i X_i)}{1 + exp(w_0 + \sum_i w_i X_i)}$$

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1|X^l, W) + (1 - Y^l) \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1|X^l, W)}{P(Y^l = 0|X^l, W)} + \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l)) \end{aligned}$$

Maximizing Conditional Log Likelihood

$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

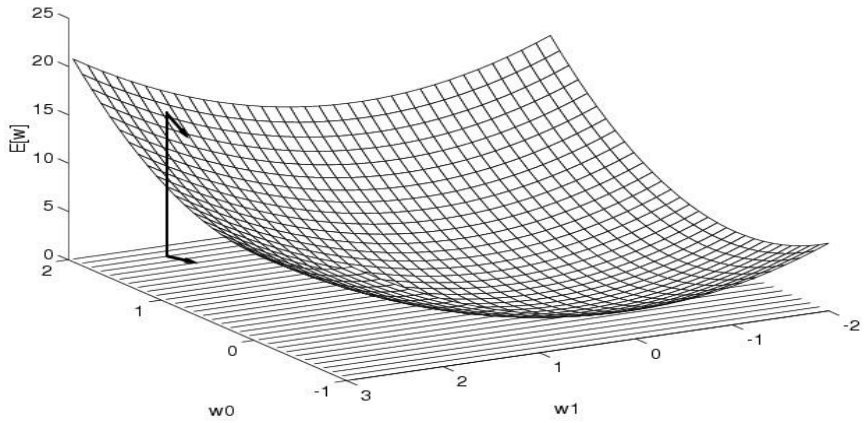
$$P(Y=1|X,W) = \frac{exp(w_0+\sum_i w_i X_i)}{1+exp(w_0+\sum_i w_i X_i)}$$

$$\begin{aligned} l(W) &\equiv \ln \prod_l P(Y^l|X^l,W) \\ &= \sum_l Y^l(w_0+\sum_i^n w_i X_i^l) - \ln(1+exp(w_0+\sum_i^n w_i X_i^l)) \end{aligned}$$

Bad news: no closed-form solution(that can be evaluated in a finite number of operations) to maximize $l(W)$

Maximizing Conditional Log Likelihood: Gradient descent algorithm

Gradient Descent



Gradient

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

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Maximize Conditional Log Likelihood: Gradient Ascent algorithm

$$l(W) \equiv \ln \prod_l P(Y^l|X^l, W)$$

$$= \sum_l Y^l(w_0 + \sum_i^n w_i X_i^l) - \ln(1 + exp(w_0 + \sum_i^n w_i X_i^l))$$

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l(Y^l - \hat{P}(Y^l = 1|X^l, W))$$

Gradient ascent algorithm: iterate until change < ε

For all i, repeat

$$w_i \leftarrow w_i + \eta \sum_l X_i^l(Y^l - \hat{P}(Y^l = 1|X^l, W))$$

MAP

- Maximum conditional likelihood estimate

$$W \leftarrow \arg\max_{W} \ln \prod_{l} P(Y^l | X^l, W)$$

$$w_i \leftarrow w_i + \eta \sum_{l} X_i^l (Y^l - \hat{P}(Y^l = 1 | X^l, W))$$

MAP estimates and Regularization

- Maximum a posteriori estimate

$$W \leftarrow \arg\max_{W} \ln \prod_{l} P(Y^l|X^l, W)$$

$$w_i \leftarrow w_i + \eta \sum_{l} X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

$$w_i \leftarrow w_i - \eta\lambda w_i + \eta \sum_{l} X_i^l (Y^l - \hat{P}(Y^l = 1|X^l, W))$$

λ is called a "regularization" term
- helps reduce overfitting
- keep weights nearer to zero
- used very frequently in Logistic Regression

Logistic regression more generally

- Logistic regression when Y not boolean (but still discrete-valued).
- Now $y \in \{y_1 \ldots y_R\}$: learn *R-1* sets of weights

for *k<R*

$$P(Y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^{n} w_{ki}X_i)}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

for *k=R*

$$P(Y = y_R|X) = \frac{1}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

Generative vs. Discriminative Classifiers

Training classifiers involves estimating f: X → Y, or P(Y|X)

Generative classifiers (e.g., Naïve Bayes)

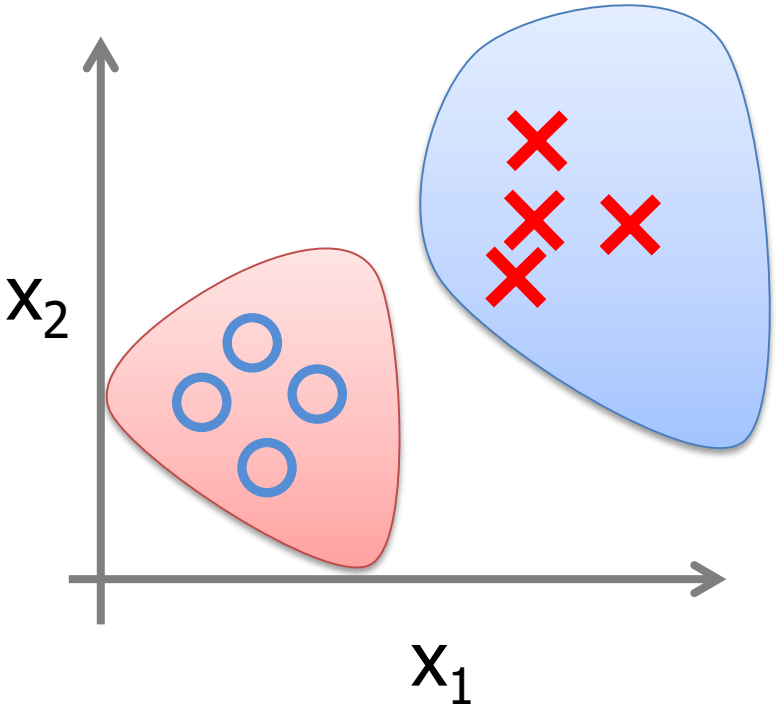
- Assume some functional form for P(X|Y), P(X)
- Estimate parameters of P(X|Y), P(X) directly from training data
- Use Bayes rule to calculate P(Y|X= x_i)

Discriminative classifiers (e.g., Logistic regression)

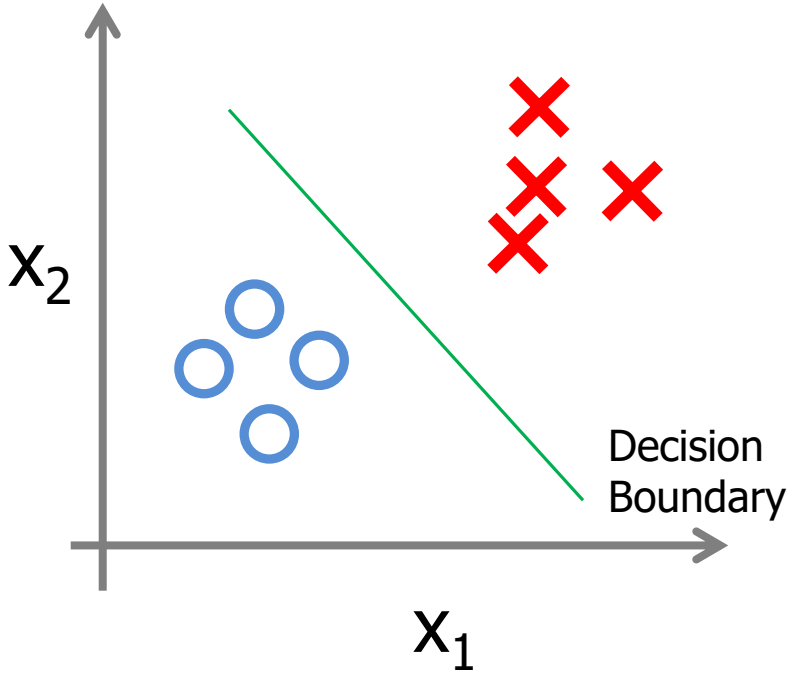
- Assume some functional form for P(Y|X)
- Estimate parameters of P(Y|X) directly from training data

Probabilistic Generative Model versus Probabilistic Discriminative Model

Generative:



Discriminative:



Naïve Bayes versus Logistic Regression

- Naïve Bayes are Generative Models
- Logistic Regression are Discriminative Models
- Naïve Bayes easy to construct
- Naive Bayes also assumes that the features are conditionally independent. Real data sets are never perfectly independent
- When the training size reaches infinity, logistic regression performs better than the generative model Naive Bayes.
  - Optional reading by Ng and Jordan has proofs and experiments

Regression

Wish to learn f: X→ Y, where Y is real, given {<x^1,y^1>…<x^n,y^n>}

- Geometric Approach
  - Least squares function fitting given {<x^1,y^1>…<x^n,y^n>}

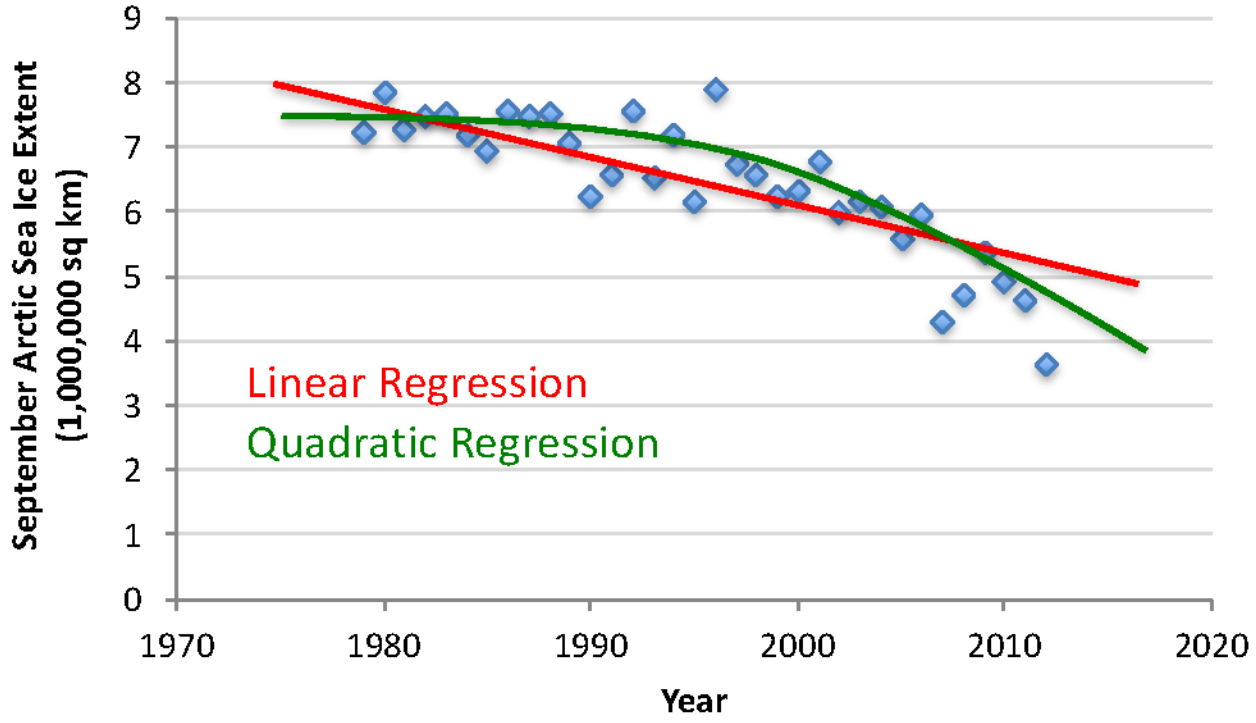
- Bayesian Approach
  - Choose some parameterized form for P(Y|X; θ)
    - (θ is the vector of parameters)

  - Derive learning algorithm as MCLE estimate for θ

Geometric Approach

Given:

- Data $\boldsymbol{X} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\right\}$ where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$
- Corresponding labels $\boldsymbol{y} = \left\{y^{(1)}, \ldots, y^{(n)}\right\}$ where $y^{(i)} \in \mathbb{R}$



Data from G. Witt. Journal of Statistics Education, Volume 21, Number 1 (2013)

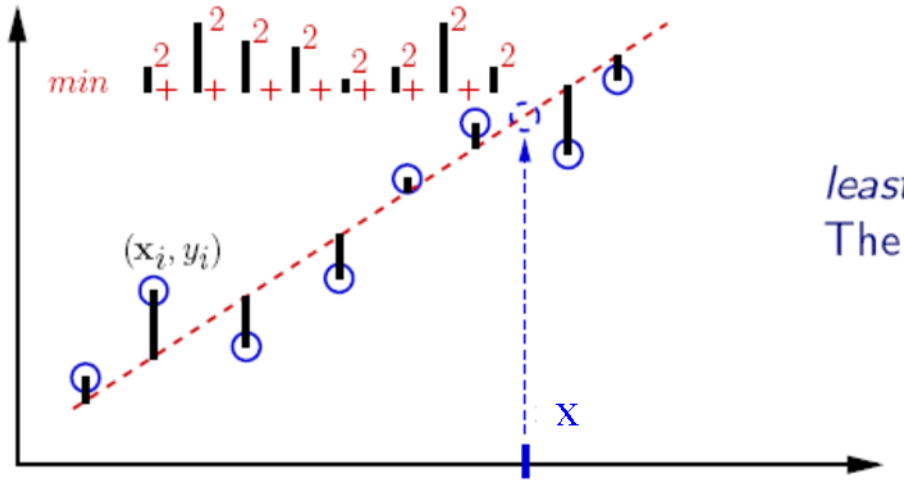
Linear Regression

- Hypothesis:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d = \sum_{j=0}^{d} \theta_j x_j$$

Assume $x_0 = 1$

- Fit model by minimizing sum of squared errors



least squares (LSQ)
The fitted line is used as a predictor

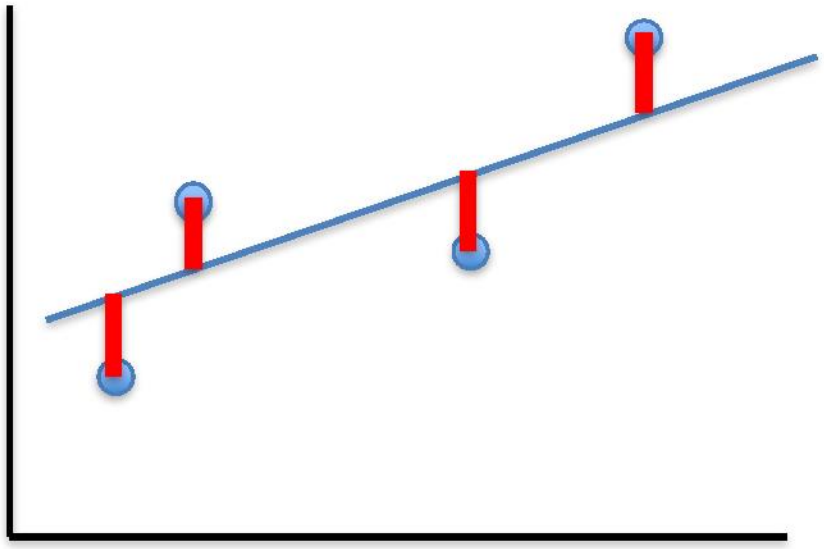
Figures are courtesy of Greg Shakhnarovich

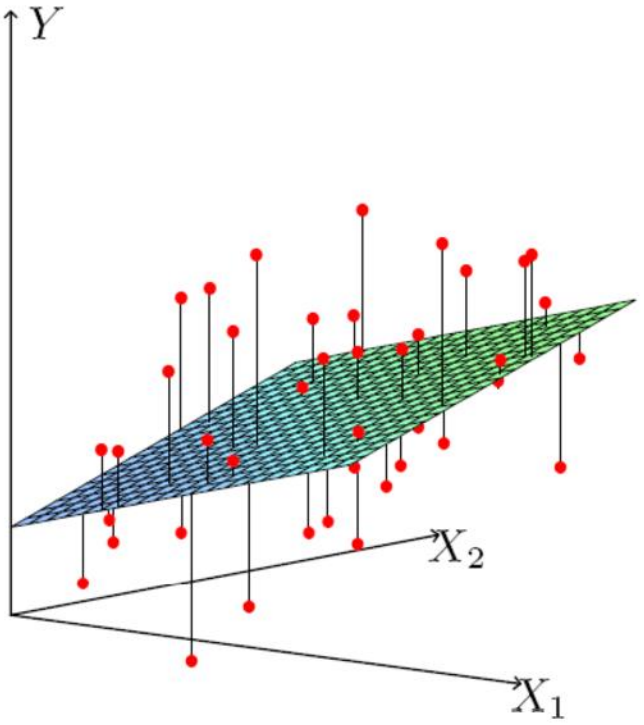
Least Squares Linear Regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$





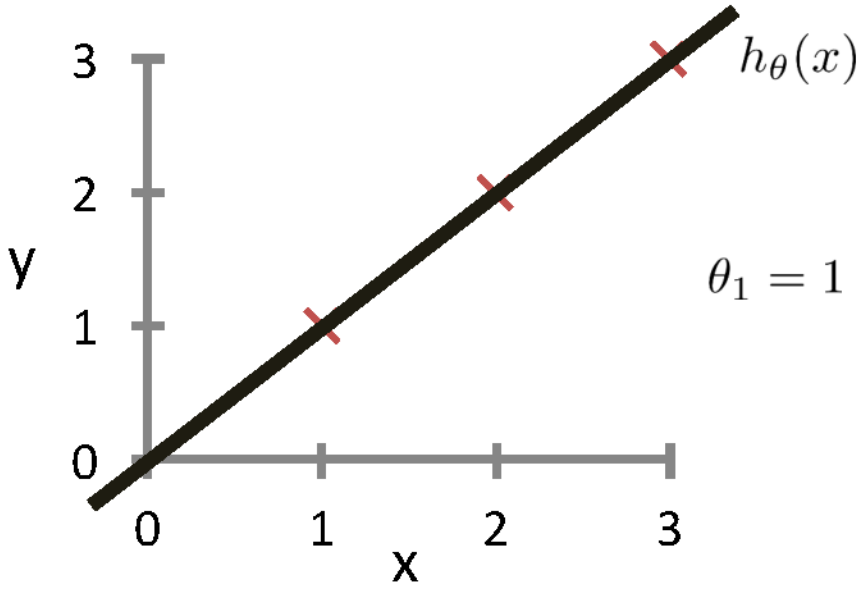
Intuition Behind Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

For insight on J(), let's assume $x \in \mathbb{R}$ so $\boldsymbol{\theta} = [\theta_0, \theta_1]$

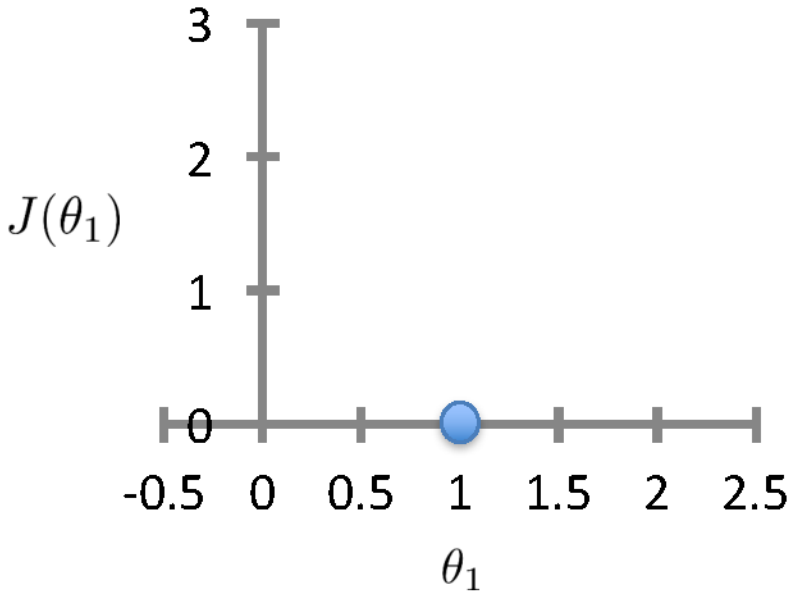
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



Based on example by Andrew Ng

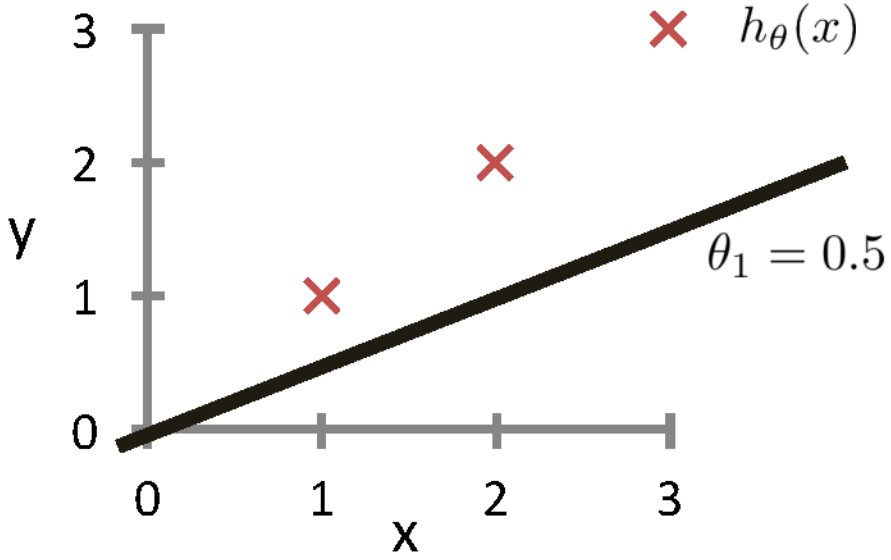
Intuition Behind Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

For insight on J(), let's assume $x \in \mathbb{R}$ so $\boldsymbol{\theta} = [\theta_0, \theta_1]$

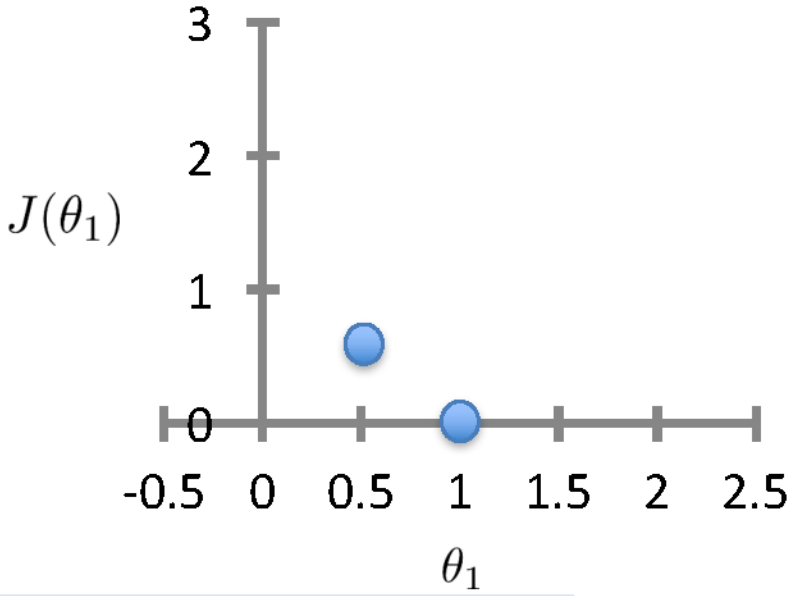
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



$$J([0, 0.5]) = \frac{1}{2 \times 3}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right] \approx 0.58$$

Based on example by Andrew Ng

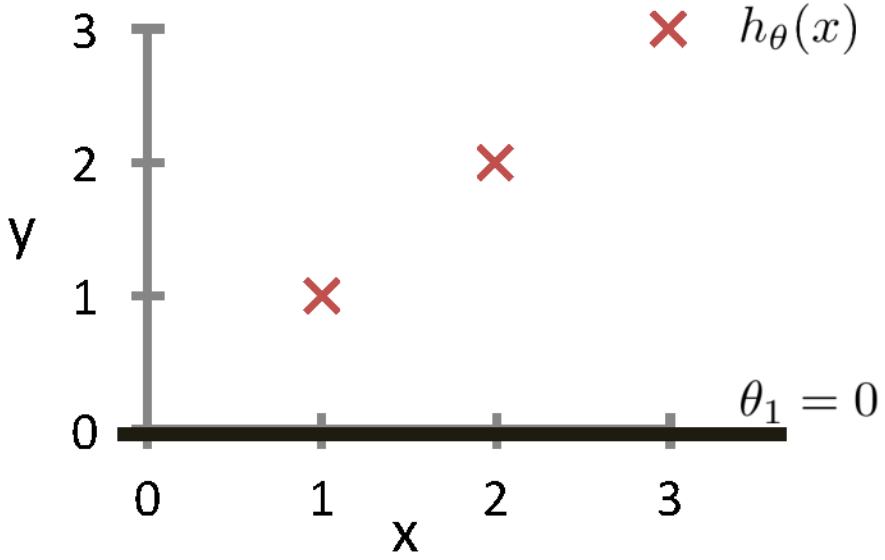
Intuition Behind Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

For insight on J(), let's assume $x \in \mathbb{R}$ so $\boldsymbol{\theta} = [\theta_0, \theta_1]$

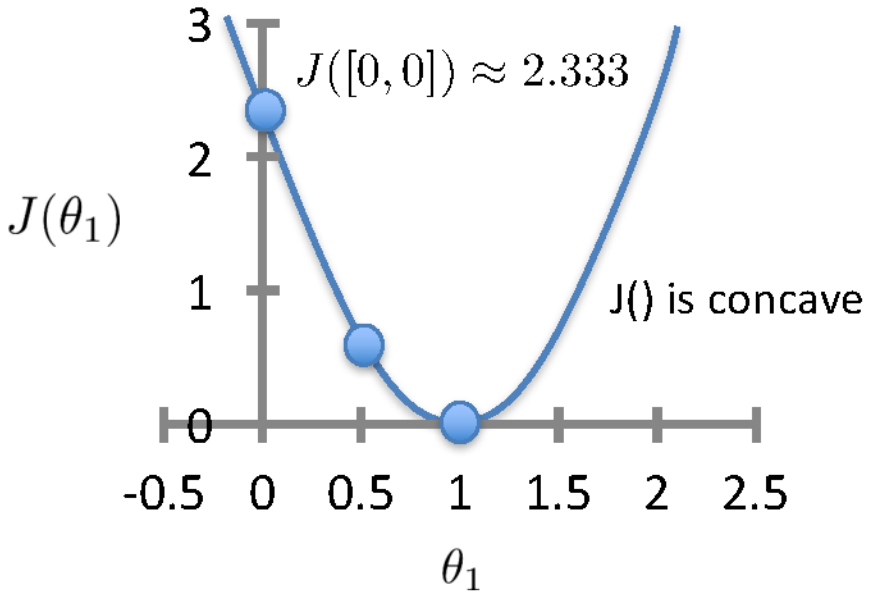
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



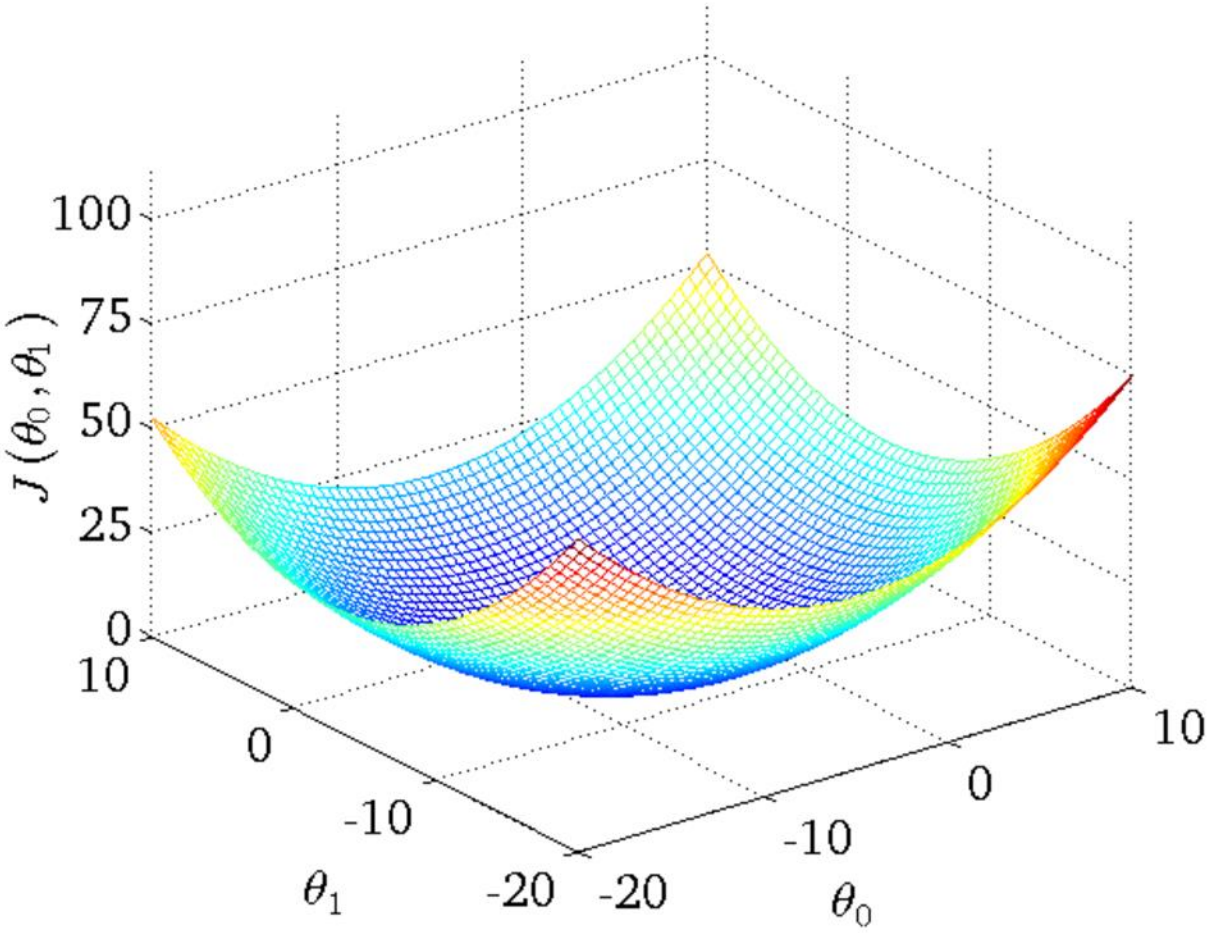
$J(\theta_1)$

(function of the parameter θ_1)



Based on example by Andrew Ng

Intuition Behind Cost Function



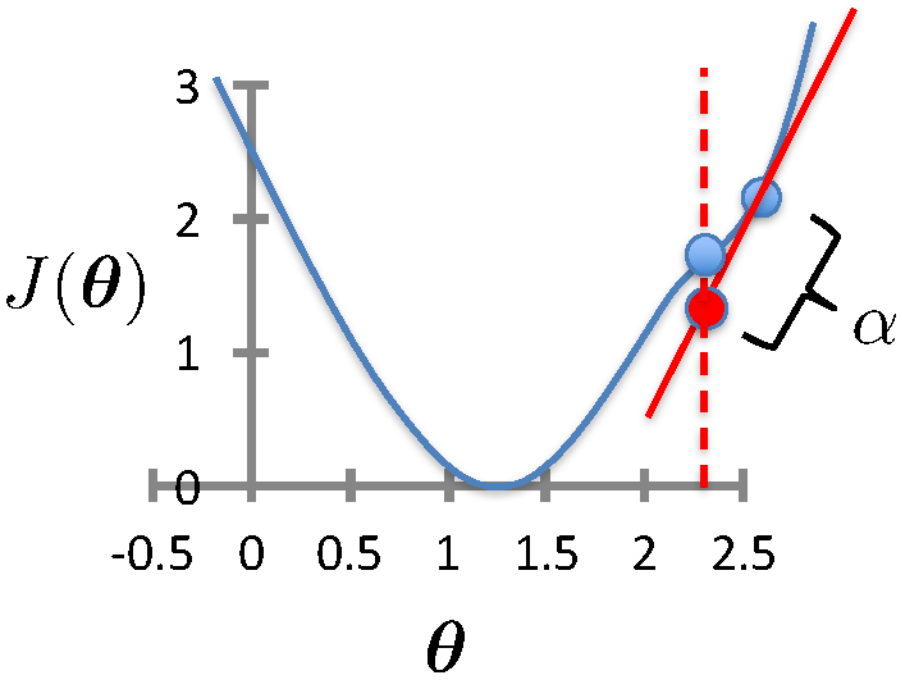
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

For Linear Regression:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 \\
&= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}
\end{aligned}$$

Gradient Descent for Linear Regression

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

simultaneous update for j = 0 ... d

- To achieve simultaneous update
  - At the start of each GD iteration, compute $h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right)$
  - Use this stored value in the update step loop

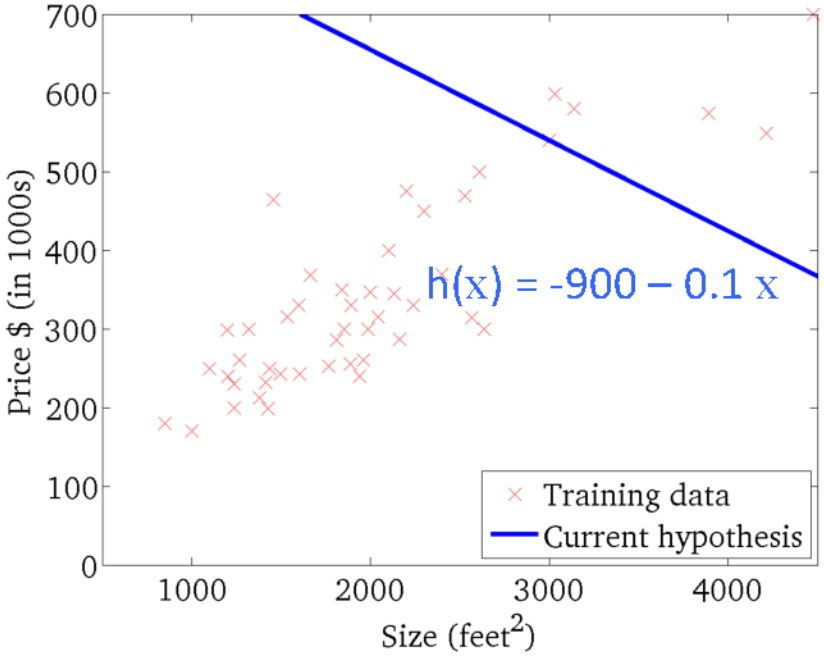
- Assume convergence when $\left\| \boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old} \right\|_2 < \epsilon$

L_2 norm: $\|\boldsymbol{v}\|_2 = \sqrt{\sum_i v_i^2} = \sqrt{v_1^2 + v_2^2 + \ldots + v_{|v|}^2}$

Gradient Descent

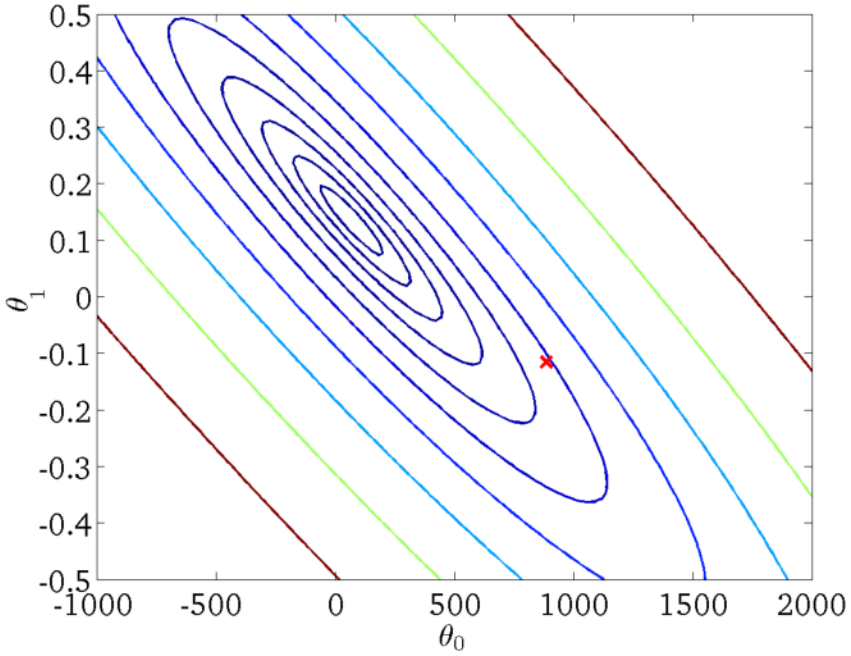
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

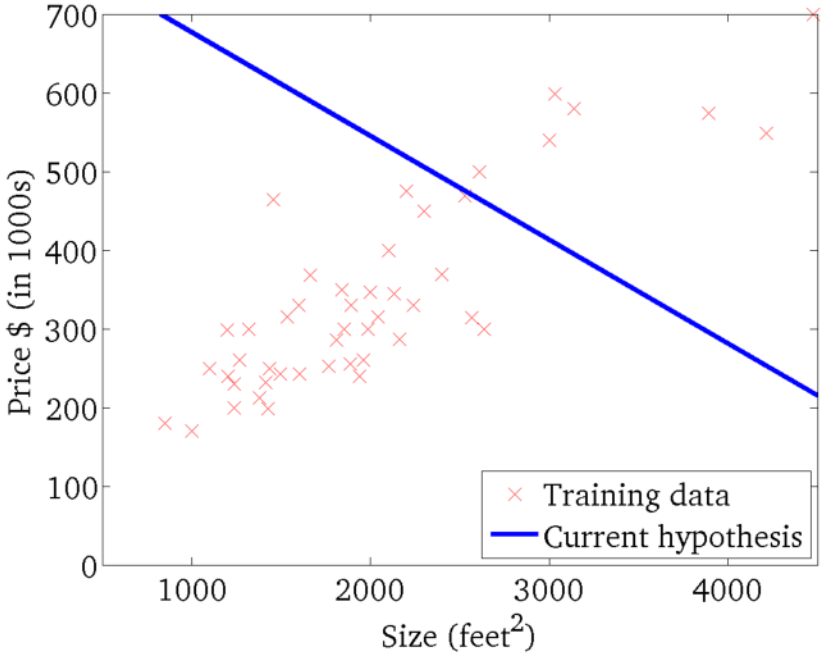
(function of the parameters θ_0, θ_1)



Gradient Descent

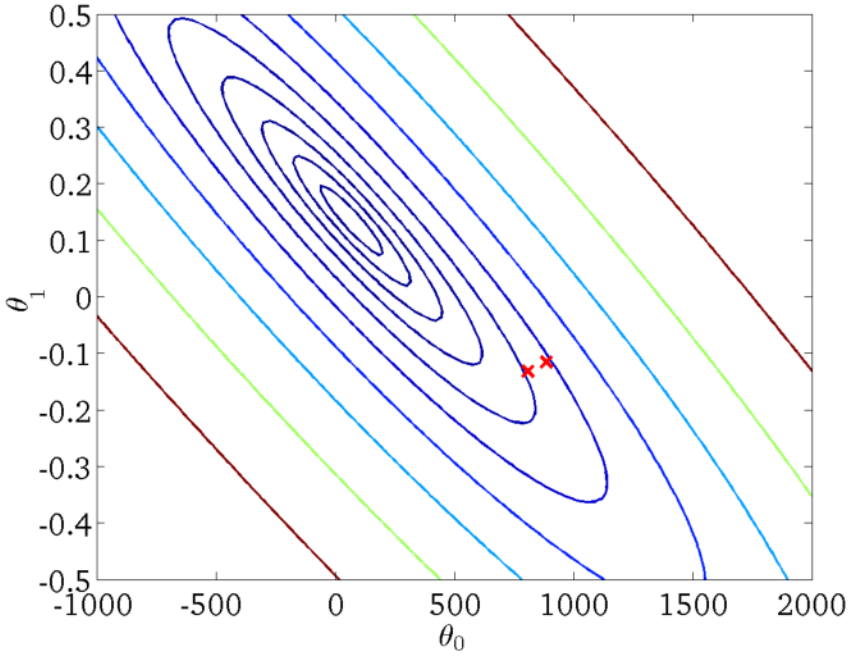
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

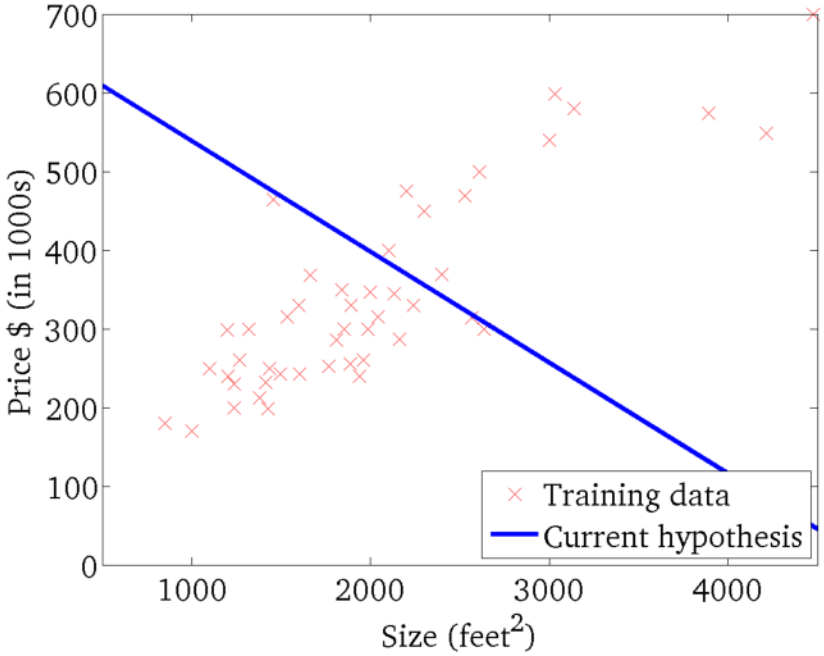
(function of the parameters θ_0, θ_1)



Gradient Descent

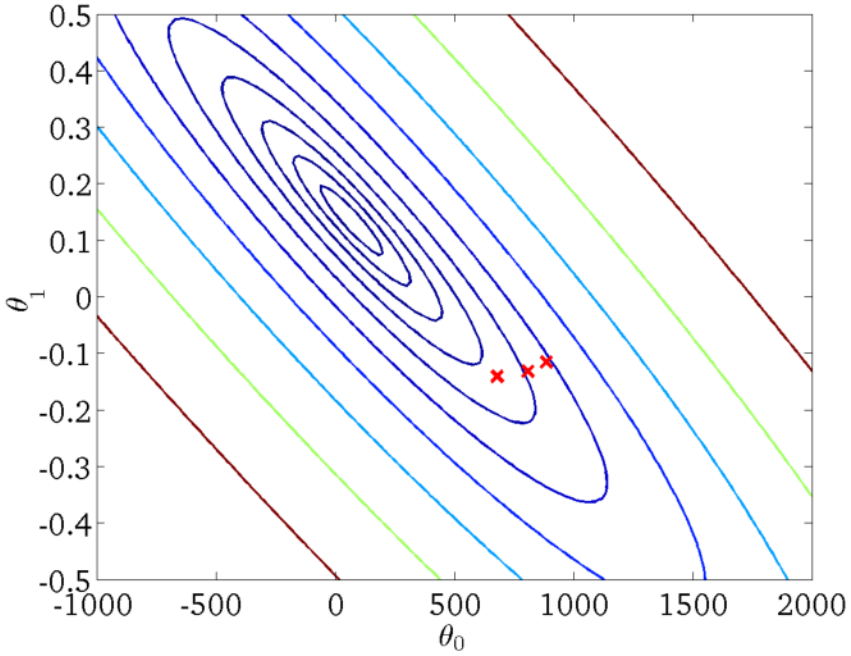
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

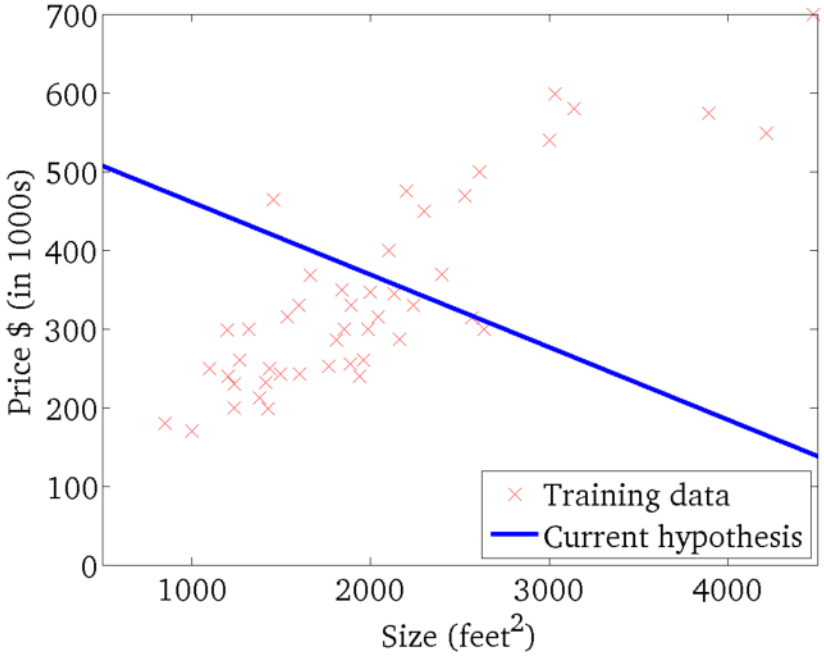
(function of the parameters θ_0, θ_1)



Gradient Descent

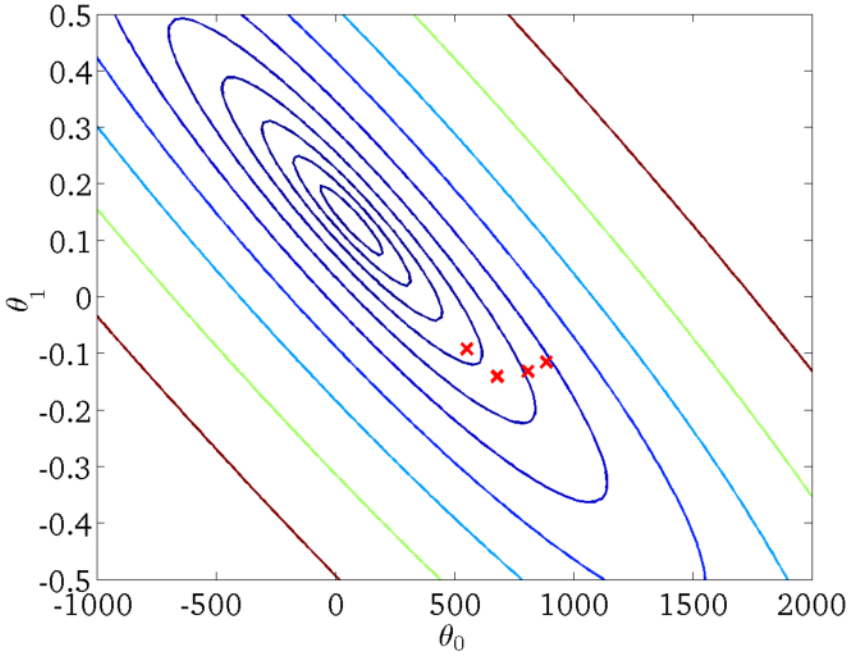
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

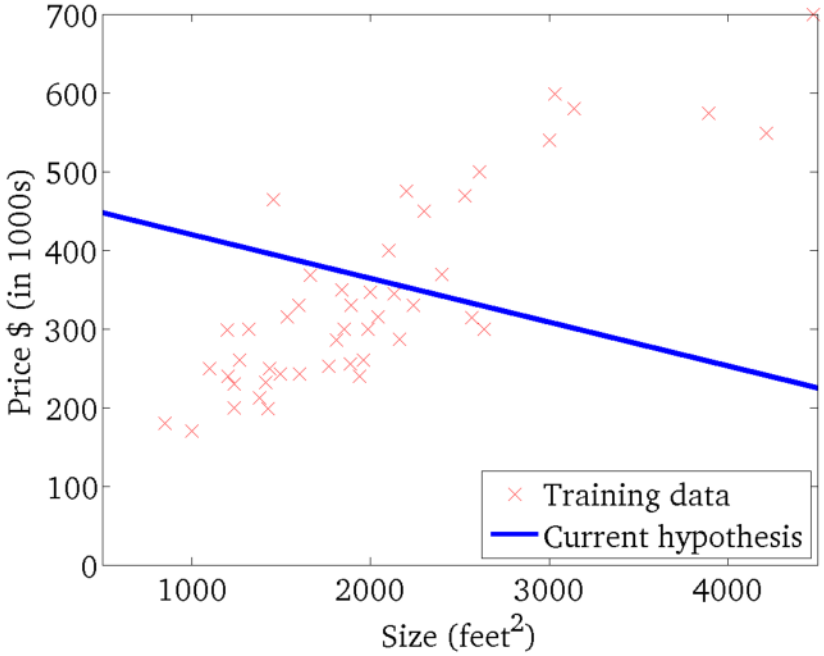
(function of the parameters θ_0, θ_1)



Gradient Descent

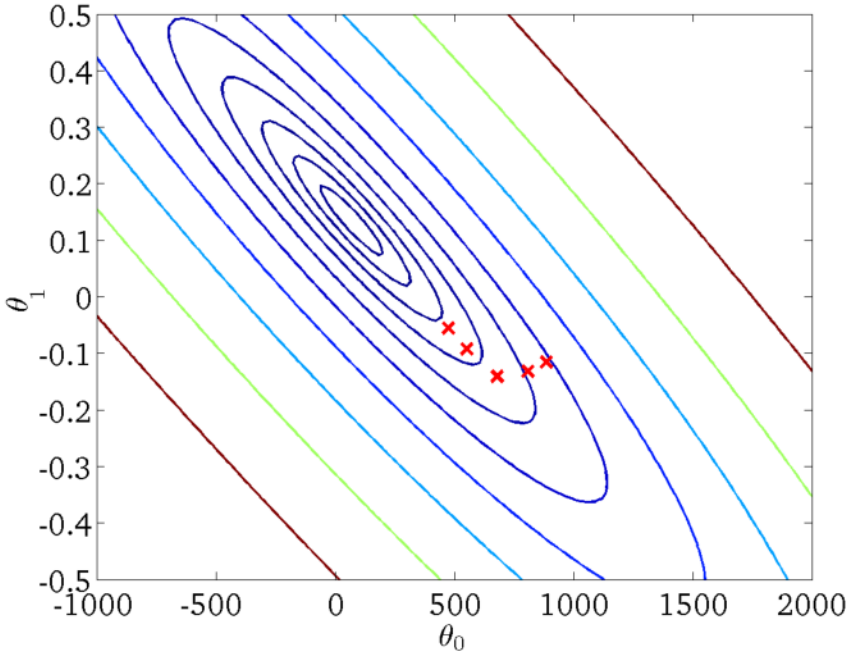
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

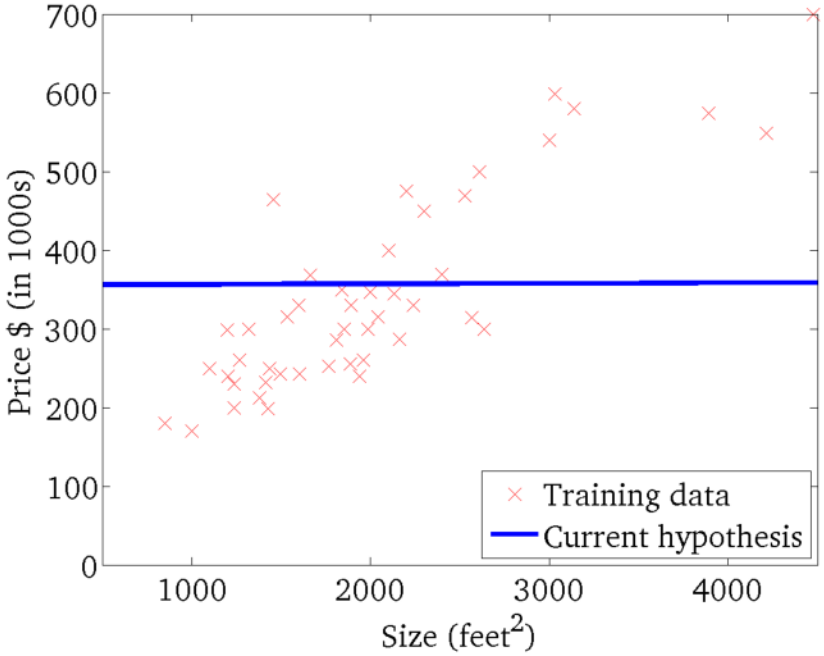
(function of the parameters θ_0, θ_1)



Gradient Descent

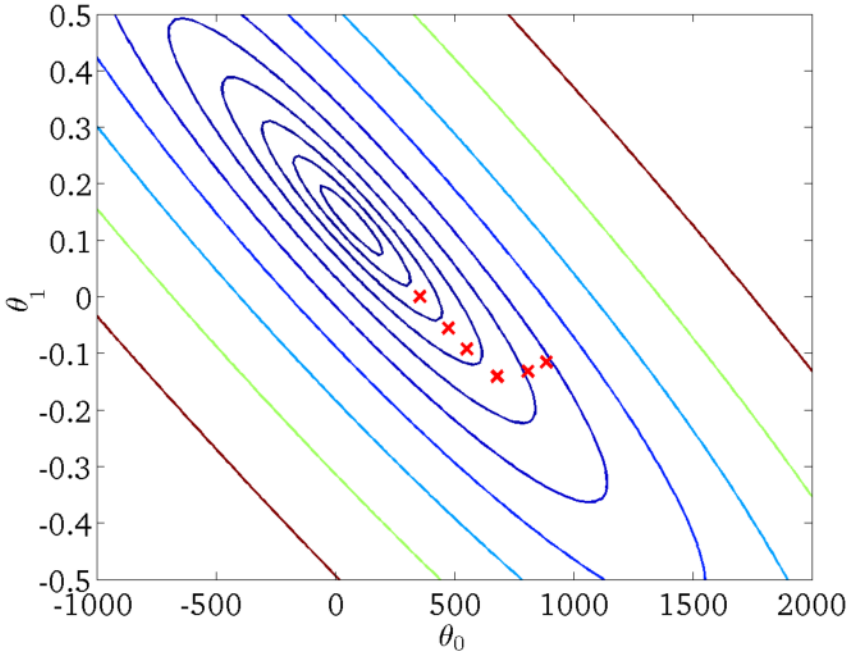
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

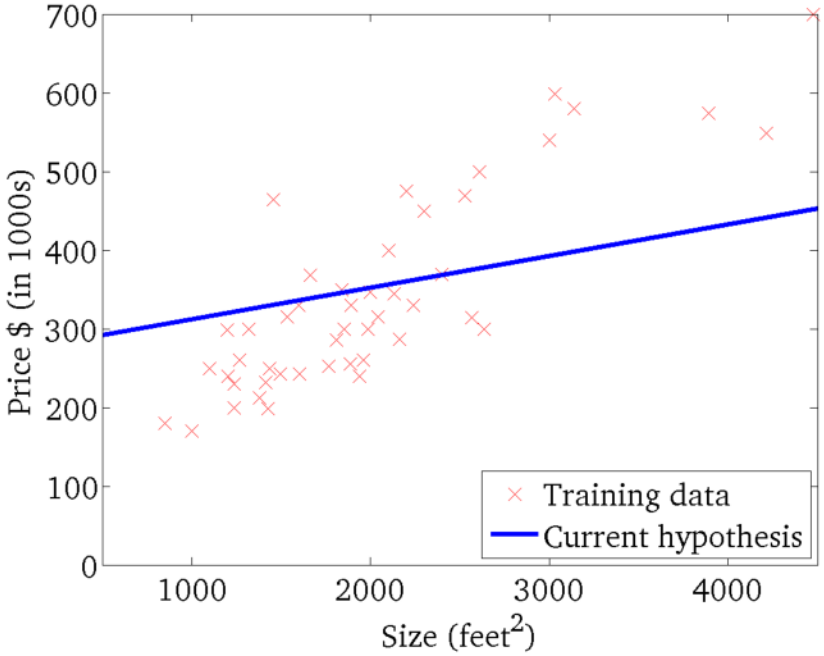
(function of the parameters θ_0, θ_1)



Gradient Descent

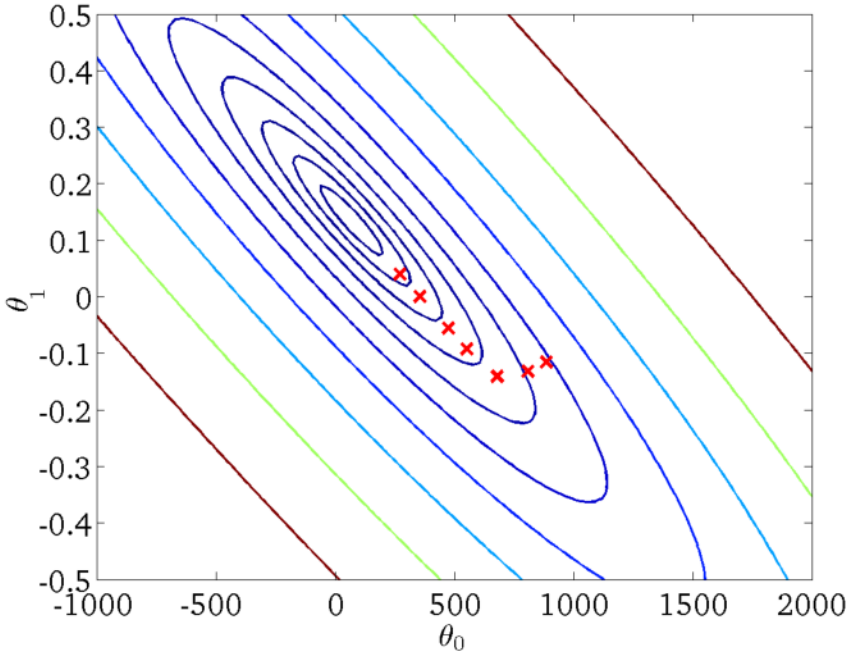
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

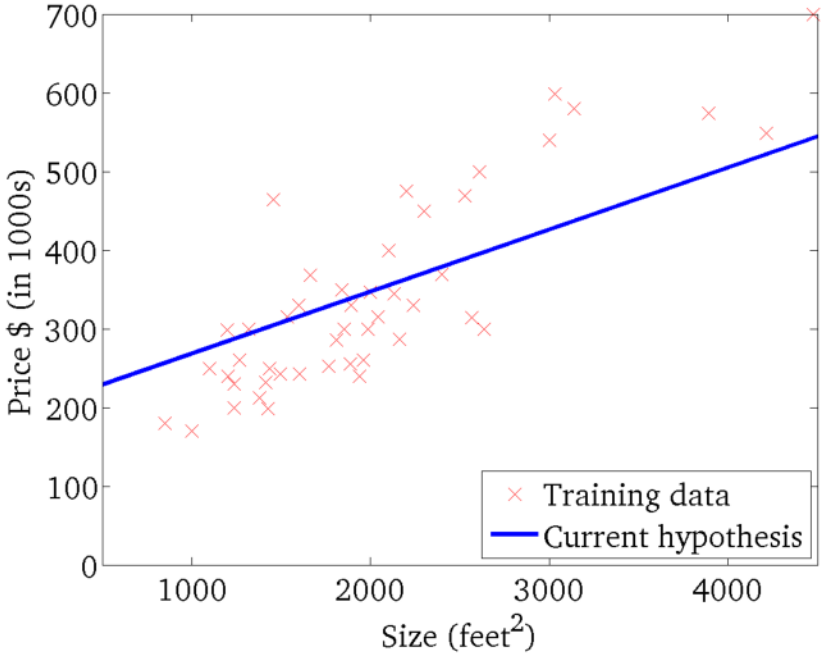
(function of the parameters θ_0, θ_1)



Gradient Descent

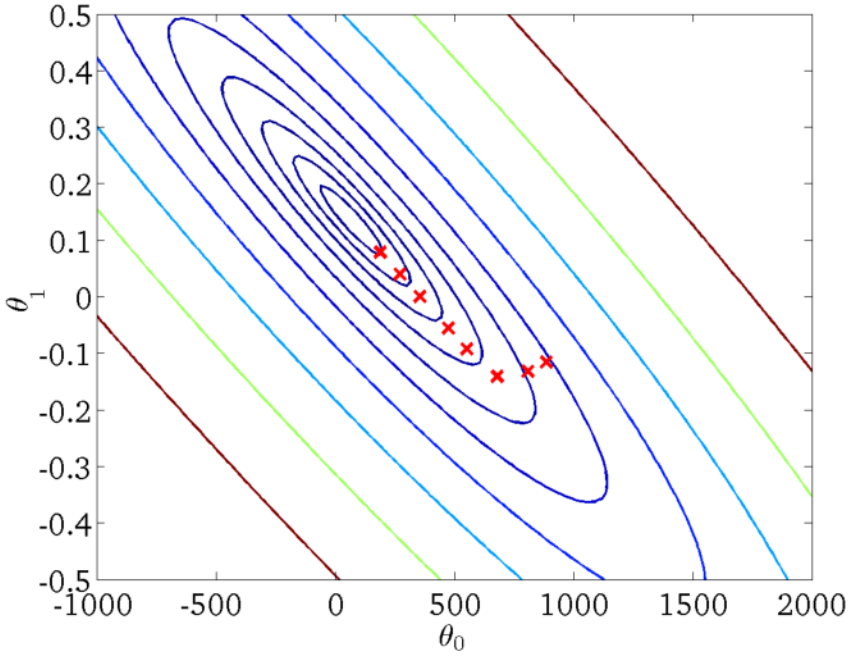
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

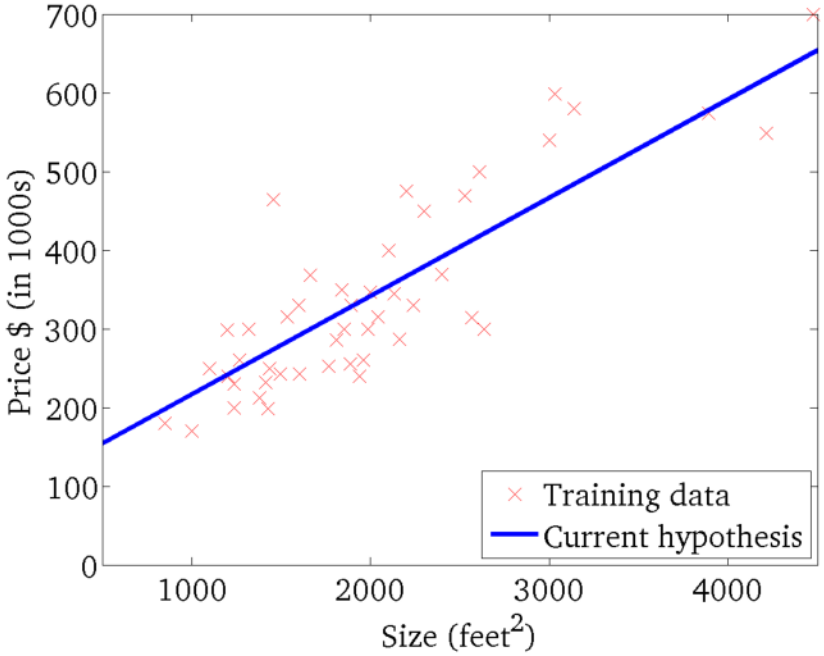
(function of the parameters θ_0, θ_1)



Gradient Descent

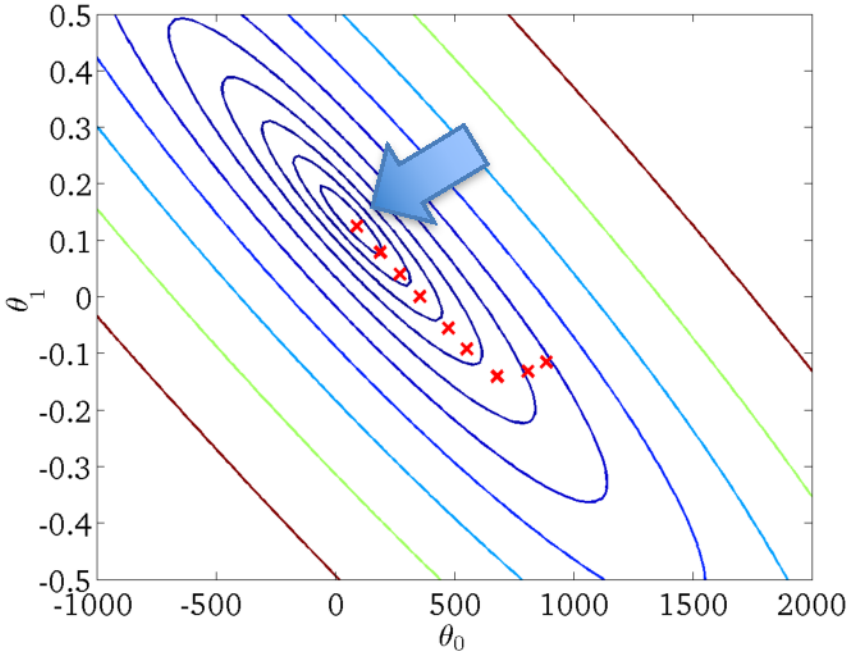
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



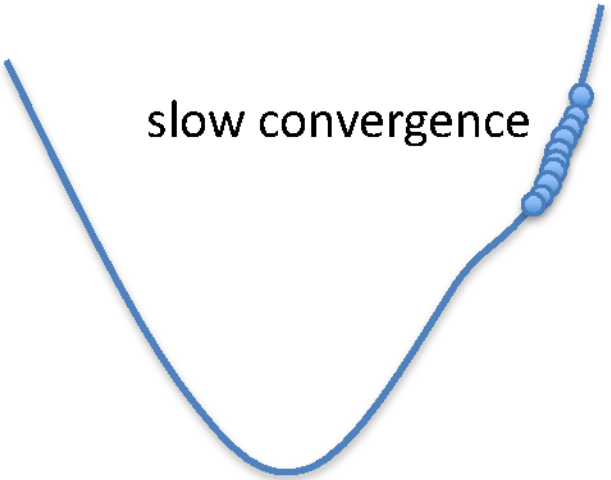
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

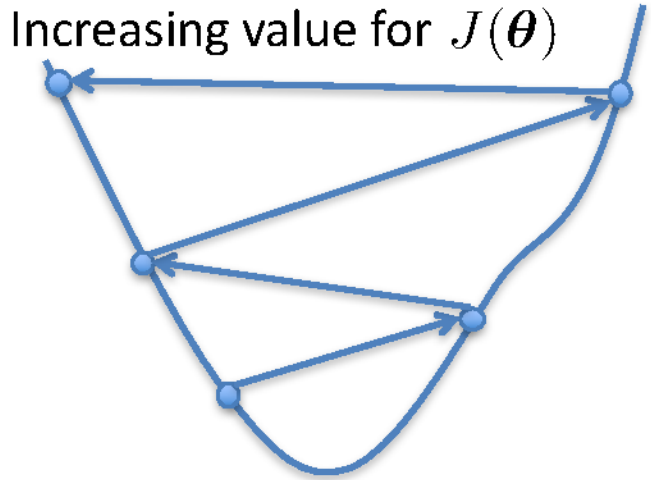


Choosing α

α too small



α too large



- May overshoot the minimum
- May fail to converge
- May even diverge

To see if gradient descent is working, print out $J(\theta)$ each iteration

- The value should decrease at each iteration
- If it doesn't, adjust α

Vectorization

- Benefits of vectorization
  - More compact equations
  - Faster code (using optimized matrix libraries)
- Consider our model:

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$

- Can write the model in vectorized form as $h(\boldsymbol{x}) = \boldsymbol{\theta}^\intercal \boldsymbol{x}$

Vectorization

- Consider our model for n instances:

$$h\left(\boldsymbol{x}^{(i)}\right) = \sum_{j=0}^{d} \theta_j x_j^{(i)}$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix}$$

$\mathbb{R}^{(d+1)\times 1}$ $\mathbb{R}^{n\times(d+1)}$

- Can write the model in vectorized form as $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{\theta}$

Vectorization

- For the linear regression cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

$$= \frac{1}{2n}\sum_{i=1}^{n}\left(\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}^{(i)} - y^{(i)}\right)^2$$

$$= \frac{1}{2n}\underbrace{(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})^{\mathsf{T}}}_{\mathbb{R}^{1\times n}}\underbrace{(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})}_{\mathbb{R}^{n\times 1}}$$

$\mathbb{R}^{n\times(d+1)}$

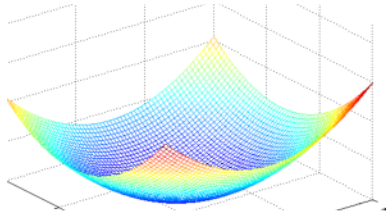
$\mathbb{R}^{(d+1)\times 1}$

Let:

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Closed Form Solution

- Instead of using GD, solve for optimal $\boldsymbol{\theta}$ analytically
  - Notice that the solution is when $\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$



- Derivation:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2n} (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})^\intercal (\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})$$

1 x 1

$$\propto \boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}^\intercal \boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y} + \boldsymbol{y}^\intercal \boldsymbol{y}$$

$$\propto \boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y} + \boldsymbol{y}^\intercal \boldsymbol{y}$$

Take derivative and set equal to 0, then solve for $\boldsymbol{\theta}$:

$$\frac{\partial}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{\theta} - 2\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y} + \cancel{\boldsymbol{y}^\intercal \boldsymbol{y}}) = 0$$

$$(\boldsymbol{X}^\intercal \boldsymbol{X})\boldsymbol{\theta} - \boldsymbol{X}^\intercal \boldsymbol{y} = 0$$

$$(\boldsymbol{X}^\intercal \boldsymbol{X})\boldsymbol{\theta} = \boldsymbol{X}^\intercal \boldsymbol{y}$$

Closed Form Solution: $\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$

Gradient Descent vs Closed Form

Gradient Descent	Closed Form Solution
• Requires multiple iterations • Need to choose α • Works well when n is large • Can support incremental learning	• Non-iterative • No need for α • Slow if n is large – Computing $(X^T X)^{-1}$ is roughly $O(n^3)$

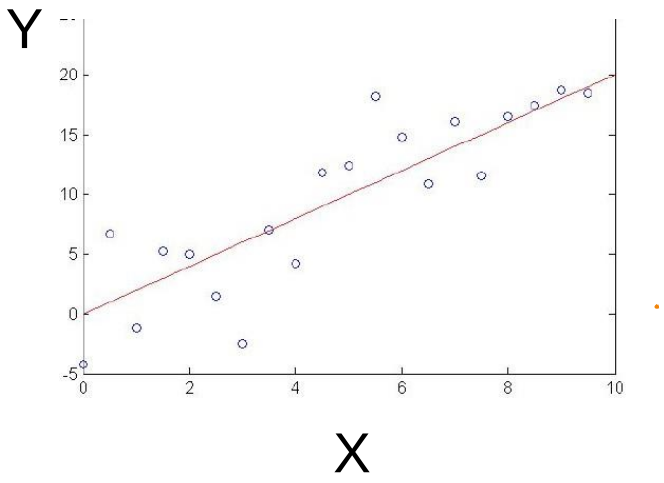
Maximum likelihood and least-squared error hypotheses

- A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution.
- Each training example is a pair of the form

(x_i, d_i) where $d_i = f(x_i) + e_i$. Here $f(x_i)$ is the noise-free value of the target function and e_i is a random variable representing the noise.

  - values of the e_i are drawn independently and that they are distributed according to a Normal distribution with zero mean

Choose parameterized form for P(Y|X; θ)



Assume Y is some deterministic f(X), plus random noise

$$y = f(x) + \epsilon \qquad \text{where} \quad \epsilon \sim N(0, \sigma)$$

Therefore Y is a random variable that follows the distribution

$$p(y|x) = N(f(x), \sigma)$$

and the expected value of y for any given x is f(x)

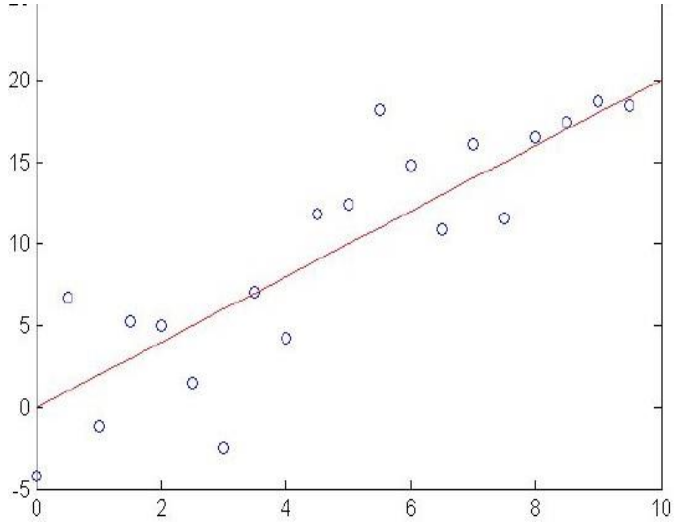
Consider Linear Regression

$$p(y|x) = N(f(x), \sigma)$$

E.g., assume f(x) is linear function of x

$$p(y|x) = N(w_0 + w_1 x, \sigma)$$

$$E[y|x] = w_0 + w_1 x$$



Notation: to make our parameters explicit, let's write

$$W =< w_0, w_1 >$$

$$p(y|x; W) = N(w_0 + w_1 x, \sigma)$$

Training Linear Regression : Maximum Conditional Likelihood Estimate (MCLE)

$$p(y|x;W) = N(w_0 + w_1 x, \sigma)$$

How can we learn W from the training data?

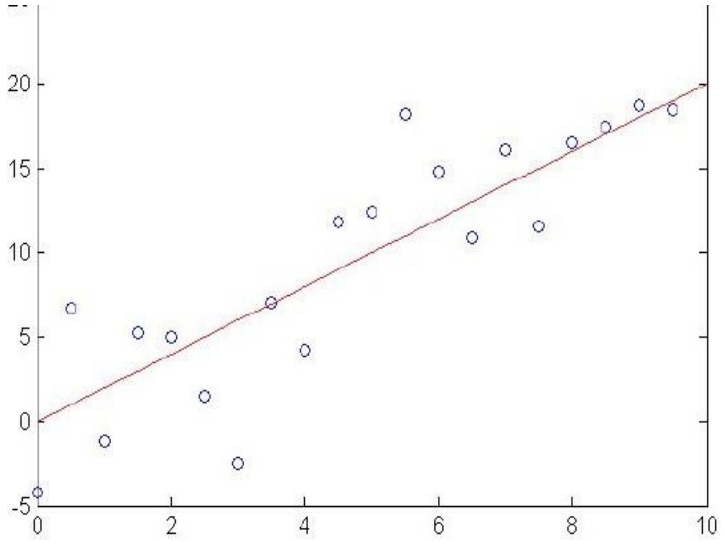
Learn Maximum Conditional Likelihood Estimate!

$$W_{MCLE} = \arg\max_W \prod_l p(y^l|x^l, W)$$

$$W_{MCLE} = \arg\max_W \sum_l \ln p(y^l|x^l, W)$$

where

$$p(y|x;W) = \frac{1}{\sqrt{2\pi\sigma^2}} \; e^{-\frac{1}{2}(\frac{y-f(x;W)}{\sigma})^2}$$



Training Linear Regression

Maximum Conditional Likelihood Estimate is equivalent to minimizing the squared error loss

$$W_{MCLE} = \arg\min_{W} \sum_{l} (y - f(x; W))^2$$

Can we derive gradient descent rule for training?

$$\frac{\partial \sum_l (y - f(x; W))^2}{\partial w_i} = \sum_l 2(y - f(x; W)) \frac{\partial (y - f(x; W))}{\partial w_i}$$

$$= \sum_l -2(y - f(x; W)) \frac{\partial f(x; W)}{\partial w_i}$$

Extending Linear Regression to More Complex Models

- The inputs **X** for linear regression can be:
  - Original quantitative inputs
  - Transformation of quantitative inputs
    - e.g. log, exp, square root, square, etc.
  - Polynomial transformation
    - example: $y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot x^3$
  - Basis expansions
  - Dummy coding of categorical inputs
  - Interactions between variables
    - example: $x_3 = x_1 \cdot x_2$

This allows use of linear regression techniques to fit non-linear datasets.

Linear Basis Function

- Simplest linear model for regression is one that involves a linear combination of the input variables

$$y(x,w)=w_0 +w_1x_1 +...+w_Dx_D$$

- Extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables, of the form

$$y(x,w)=w_0 + \sum_{j=1}^{M-1} w_j\varphi_j(x)$$

  - where $\varphi_j(x)$ are known as basis functions.
  - By denoting the maximum value of the index j by M −1, the total number of parameters in this model will be M.

Linear Basis Function

- Convenient to define an additional dummy 'basis function' $\phi_0(x)=1$. So,

$$y(x,w) = \sum_{j=1}^{M-1} w_j \varphi_j(x) = \mathbf{w}^T \boldsymbol{\phi}_j(\mathbf{x})$$

where $\mathbf{w} = (w_0,\ldots,w_{M-1})^T$ and $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots\ldots \phi_n)$

- If the original variables comprise the vector x, then the features can be expressed in terms of the basis functions {ϕj(x)}

Linear Basis Function Models

- Generally,

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j \phi_j(\boldsymbol{x})$$

basis function

- Typically, $\phi_0(\boldsymbol{x}) = 1$ so that θ_0 acts as a bias
- In the simplest case, we use linear basis functions :

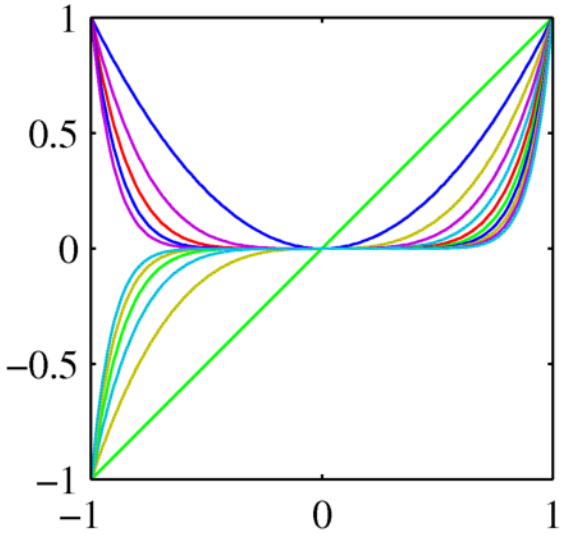
$$\phi_j(\boldsymbol{x}) = x_j$$

Linear Basis Function Models

- Polynomial basis functions:

$$\phi_j(x) = x^j$$

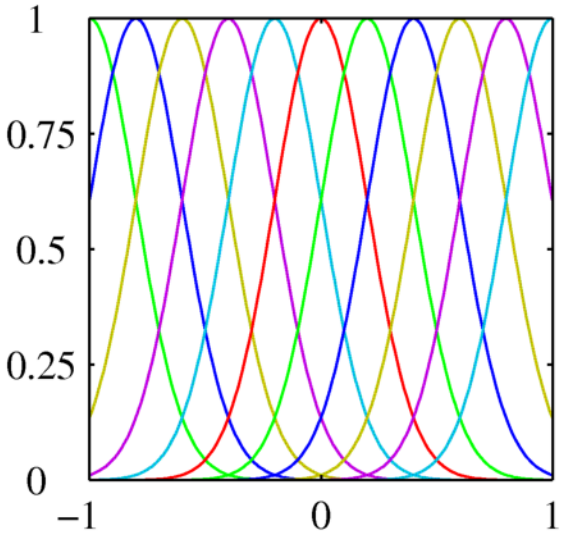
  - These are global; a small change in x affects all basis functions



- Gaussian basis functions:

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

  - These are local; a small change in x only affect nearby basis functions. μ_j and s control location and scale (width).

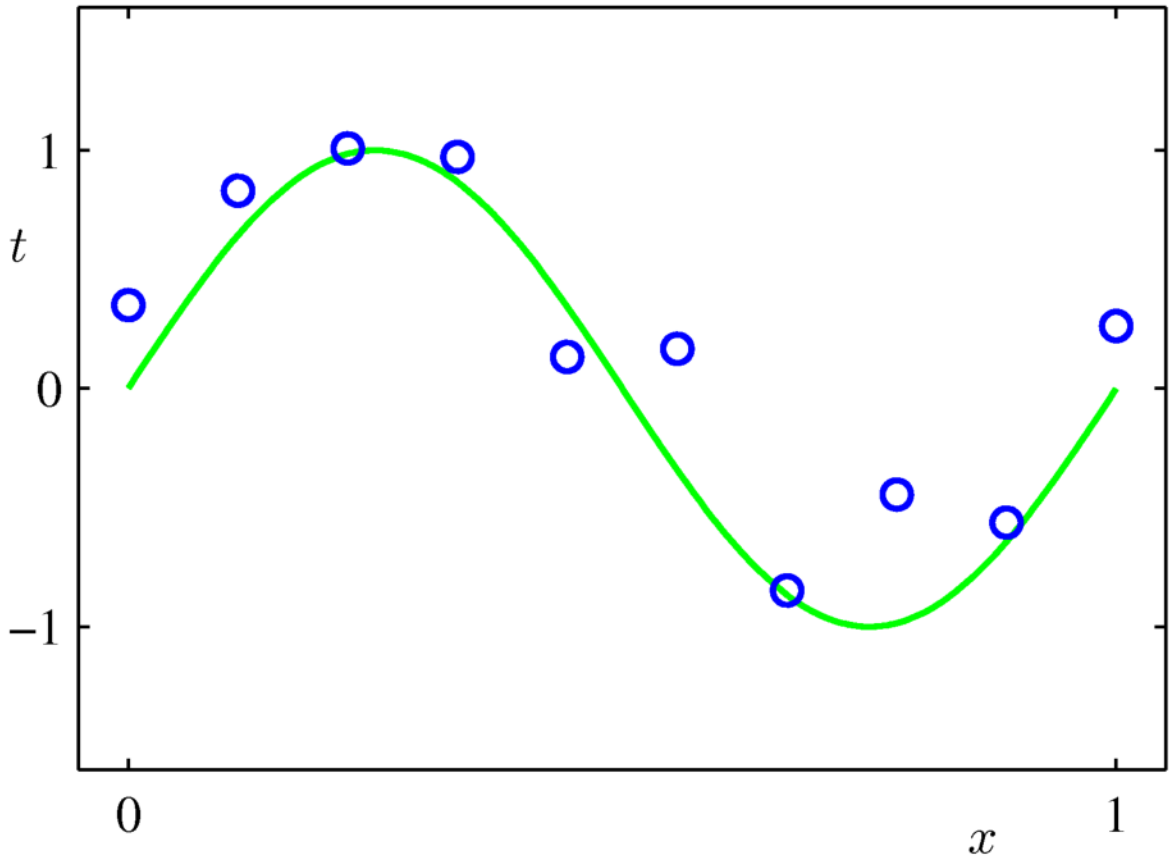


Linear Basis Function Models

- Basic Linear Model: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$
- Generalized Linear Model: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j \phi_j(\boldsymbol{x})$
- Once we have replaced the data by the outputs of the basis functions, fitting the generalized model is exactly the same problem as fitting the basic model
  - Unless we use the kernel trick – more on that when we cover support vector machines
  - Therefore, there is no point in cluttering the math with basis functions

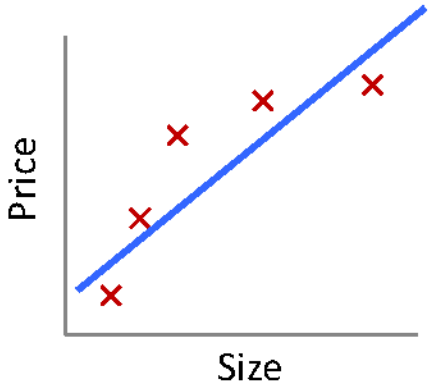
Based on slide by Geoff Hinton

Example of Fitting a Polynomial Curve with a Linear Model



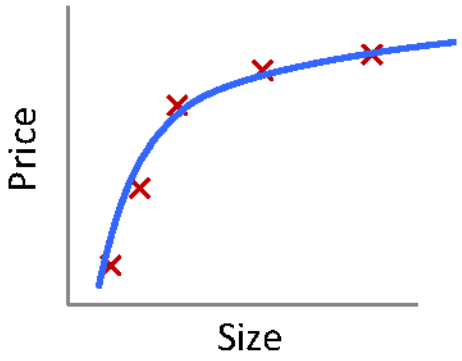
$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_p x^p = \sum_{j=0}^{p} \theta_j x^j$$

Quality of Fit



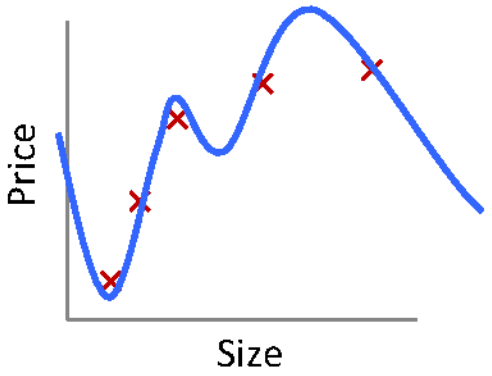
$\theta_0 + \theta_1 x$

Underfitting
(high bias)



$\theta_0 + \theta_1 x + \theta_2 x^2$

Correct fit



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfitting
(high variance)

Overfitting:

- The learned hypothesis may fit the training set very well ($J(\boldsymbol{\theta}) \approx 0$)
- ...but fails to generalize to new examples

Bias – Variance decomposition of error

$$E_{D,\varepsilon}\left\{\left(f(x)+\varepsilon-h_D(x)\right)^2\right\}$$

dataset and noise

true function

noise

learned from D

Fix test case *x*, then do this experiment:

1. Draw size *n* sample D=$(x_1,y_1),\ldots.(x_n,y_n)$
2. Train linear regressor h_D *using* D
3. Draw one test example $(x, f(x)+\varepsilon)$
4. Measure squared error of h_D on that example *x*
5. What's the expected error?

Bias – Variance decomposition of error

Notation - to simplify this

$$f \equiv f(x) + \varepsilon \qquad \hat{y} = \hat{y}_D \equiv h_D(x)$$

$$E_{D,\varepsilon}\left\{\left(f(x)+\varepsilon-h_D(x)\right)^2\right\}$$

dataset and noise

true function

noise

learned from D

$$h \equiv E_D\{h_D(x)\}$$

long-term expectation of learner's prediction on this *x* averaged over many data sets *D*

Bias – Variance decomposition of error

$$h = E_D\{\hat{y}_D\}$$

$$E_{D,\varepsilon}\left\{(f-\hat{y})^2\right\}$$

$$= E\left\{\left([f-h]+[h-\hat{y}]\right)^2\right\}$$

$$= E\left\{[f-h]^2+[h-\hat{y}]^2+2[f-h][h-\hat{y}]\right\}$$

$$= E\left\{[f-h]^2+[h-\hat{y}]^2+2[fh-f\hat{y}-h^2+h\hat{y}]\right\}$$

$$= E[(f-h)^2]+E[(h-\hat{y})^2]+2\left(E[fh]-E[f\hat{y}]-E[h^2]+E[h\hat{y}]\right)$$

$$\hat{y} = \hat{y}_D \equiv h_D(x)$$

$$f \equiv f(x)+\varepsilon$$

Bias – Variance decomposition of error

$$h = E_D\{\hat{y}_D\}$$

$$E_{D,\varepsilon}\left\{(f-\hat{y})^2\right\}$$

$$= E\left\{\left([f-h]+[h-\hat{y}]\right)^2\right\}$$

$$= E\left\{[f-h]^2+[h-\hat{y}]^2+2[f-h][h-\hat{y}]\right\}$$

$$= E\left\{[f-h]^2+[h-\hat{y}]^2+2[fh-f\hat{y}-h^2+h\hat{y}]\right\}$$

$$= E[(f-h)^2]+E[(h-\hat{y})^2]+2\left(E[fh]-E[f\hat{y}]-E[h^2]+E[h\hat{y}]\right)$$

$$\hat{y} = \hat{y}_D \equiv h_D(x)$$

$$f \equiv f(x)+\varepsilon$$

Bias – Variance decomposition of error

$h = E_D\{\hat{y}_D\}$ $\quad \hat{y} = \hat{y}_D$ $\quad f \equiv f(x) + \varepsilon$

$$
\begin{aligned}
& 2(E[fh] - E[f\hat{y}] - E[h^2] + E[h\hat{y}]) \\
&= E[f]E[h] - E[f]E[\hat{y}] - E[h]E[h] + E[h]E[\hat{y}] \\
&= E[f]E[E[\hat{y}]] - E[f]E[\hat{y}] - E[h]E[E[\hat{y}]] + E[h]E[\hat{y}] \\
&= E[f]E[\hat{y}] - E[f]E[\hat{y}] - E[h]E[\hat{y}] + E[h]E[\hat{y}] \\
&= 0
\end{aligned}
$$

Bias – Variance decomposition of error

$$E_{D,\varepsilon}\left\{(f-\hat{y})^2\right\}$$
$$= E\left\{\left([f-h]+[h-\hat{y}]\right)\right\}^2$$
$$= E\left\{[f-h]^2+[h-\hat{y}]^2+2[f-h][h-\hat{y}]\right\}$$
$$= E[(f-h)^2]+E[(h-\hat{y})^2]$$

$$h \equiv E_D\{h_D(x)\}$$
$$\hat{y} = \hat{y}_D \equiv h_D(x)$$
$$f \equiv f(x) + \varepsilon$$

BIAS^2

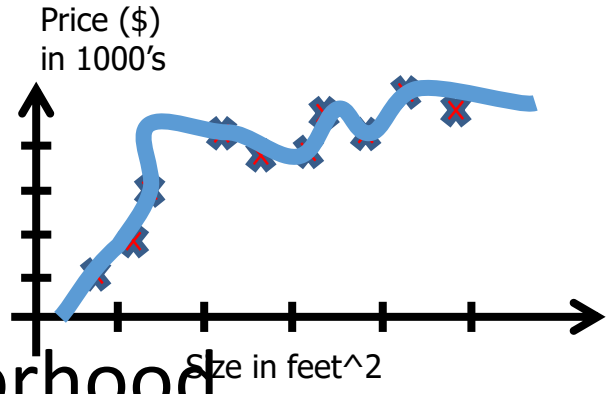
Squared difference between best possible prediction for x, f(x), and our "long-term" expectation for what the learner will do if we averaged over many datasets D, $E_D[h_D(x)]$

VARIANCE

Squared difference btwn our long-term expectation for the learners performance, $E_D[h_D(x)]$, and what we expect in a representative run on a dataset D (hat y)

Addressing overfitting

- $x_1 =$ size of house
- $x_2 =$ no. of bedrooms
- $x_3 =$ no. of floors
- $x_4 =$ age of house
- $x_5 =$ average income in neighborhood
- $x_6 =$ kitchen size
- $\vdots$
- x_{100}



Slide credit: Andrew Ng

Addressing overfitting

- **1. Reduce number of features.**
  - Manually select which features to keep.
  - Model selection algorithm

- **2. Regularization.**
  - Keep all the features, but reduce magnitude/values of parameters θ_j.
  - Works well when we have a lot of features, each of which contributes a bit to predicting y.

Slide credit: Andrew Ng

Regularization

- A method for automatically controlling the complexity of the learned hypothesis
- **Idea**: penalize for large values of θ_j
  - Can incorporate into the cost function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label
- Can also address overfitting by eliminating features (either manually or via model selection)

Regularization

- Linear regression objective function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

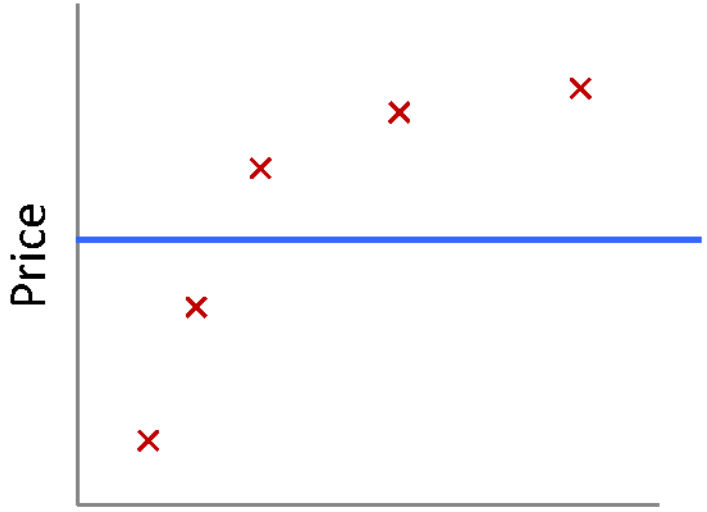
model fit to data — regularization

  - λ is the regularization parameter ($\lambda \geq 0$)
  - No regularization on θ_0!

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

0 0 0 0

Based on example by Andrew Ng

Regularized Linear Regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Gradient update:

$\frac{\partial}{\partial\theta_0}J(\theta)$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$\frac{\partial}{\partial\theta_j}J(\theta)$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} - \lambda\theta_j$$

regularization

Regularized Linear Regression

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} - \lambda\theta_j$$

- We can rewrite the gradient step as:

$$\theta_j \leftarrow \theta_j\left(1 - \alpha\lambda\right) - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Decision Tree and Random Forest

Entropy in general

- Entropy measures the amount of information in a random variable

$$H(X) = -p_+ \log_2 p_+ - p_- \log_2 p_- \qquad X = \{+, -\}$$

for binary classification [two-valued random variable]

$$H(X) = -\sum_{i=1}^{c} p_i \log_2 p_i = \sum_{i=1}^{c} p_i \log_2 1/p_i \qquad X = \{i, \ldots, c\}$$

for classification in *c* classes

Entropy in binary classification

- Entropy measures the *impurity* of a collection of examples. It depends from the distribution of the random variable *p.*
  - S is a collection of training examples
  - p_+ the proportion of positive examples in S
  - p_- the proportion of negative examples in S

$$Entropy\,(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_- \quad [0 \log_2 0 = 0]$$

$$Entropy\,([14+, 0-]) = -14/14 \log_2 (14/14) - 0 \log_2 (0) = 0$$

$$Entropy\,([9+, 5-]) = -9/14 \log_2 (9/14) - 5/14 \log_2 (5/14) = 0.94$$

$$Entropy\,([7+, 7-]) = -7/14 \log_2 (7/14) - 7/14 \log_2 (7/14) =$$

$$= 1/2 + 1/2 = 1 \qquad [\log_2 1/2 = -1]$$

Note: the log of a number < 1 is negative, $0 \le p \le 1$, $0 \le entropy \le 1$

- https://www.easycalculation.com/log-base2-calculator.php

Information gain as entropy reduction

- *Information gain* is the *expected* reduction in entropy caused by partitioning the examples on an attribute.
- The higher the information gain the more effective the attribute in classifying training data.
- Expected reduction in entropy knowing *A*

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|Sv|}{|S|} Entropy(S_v)$$

$Values(A)$ possible values for A
S_v subset of S for which A has value v

Example: Information gain

- Let
  - $Values(Wind) = \{Weak, Strong\}$
  - $S = [9+, 5-]$
  - $S_{Weak} = [6+, 2-]$
  - $S_{Strong} = [3+, 3-]$
- Information gain due to knowing *Wind*:

$$Gain(S, Wind) = Entropy(S) - 8/14\ Entropy(S_{Weak}) - 6/14\ Entropy(S_{Strong})$$

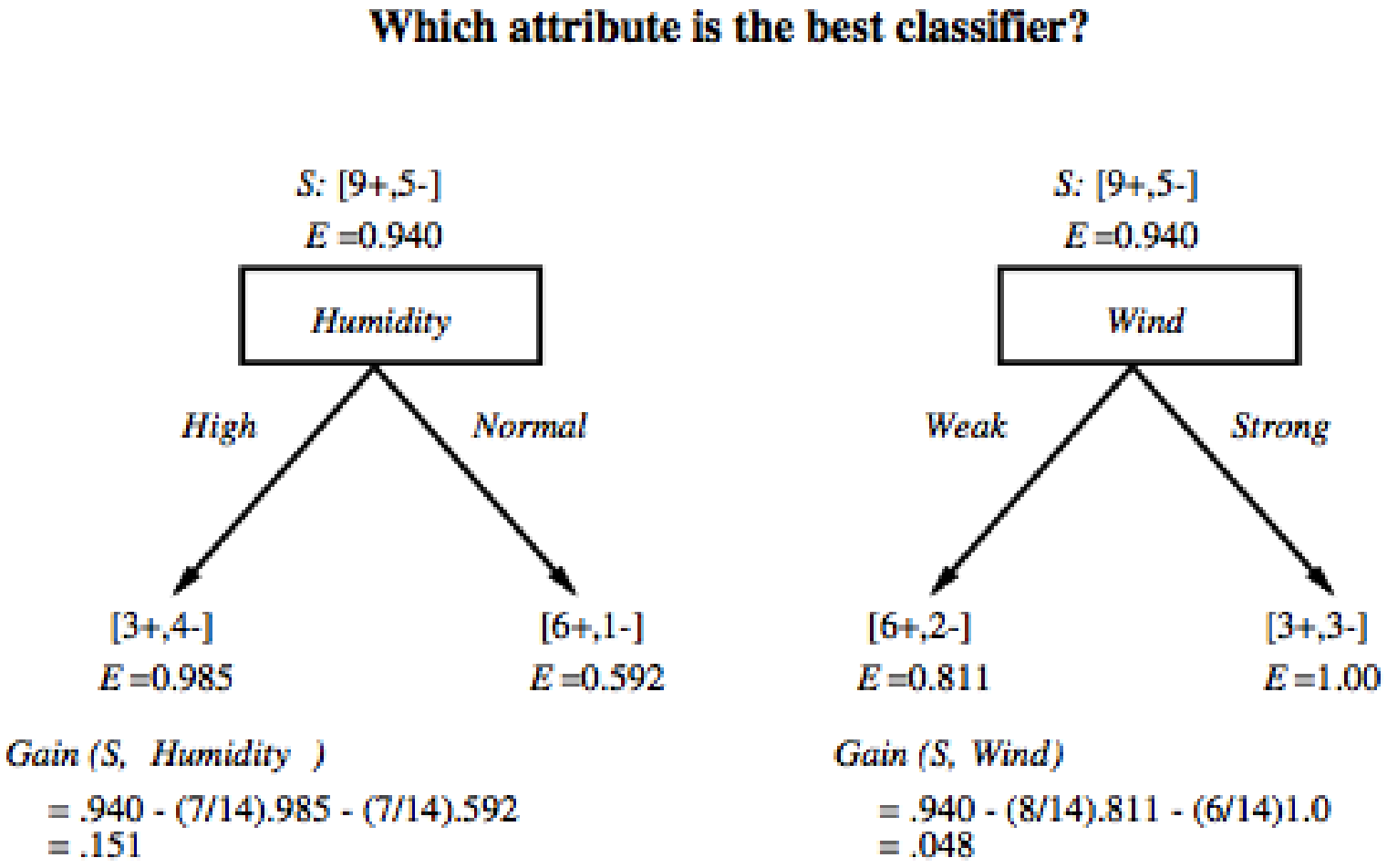
$$= 0.94 - 8/14 \times 0.811 - 6/14 \times 1.00$$

$$= 0.048$$

Example

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

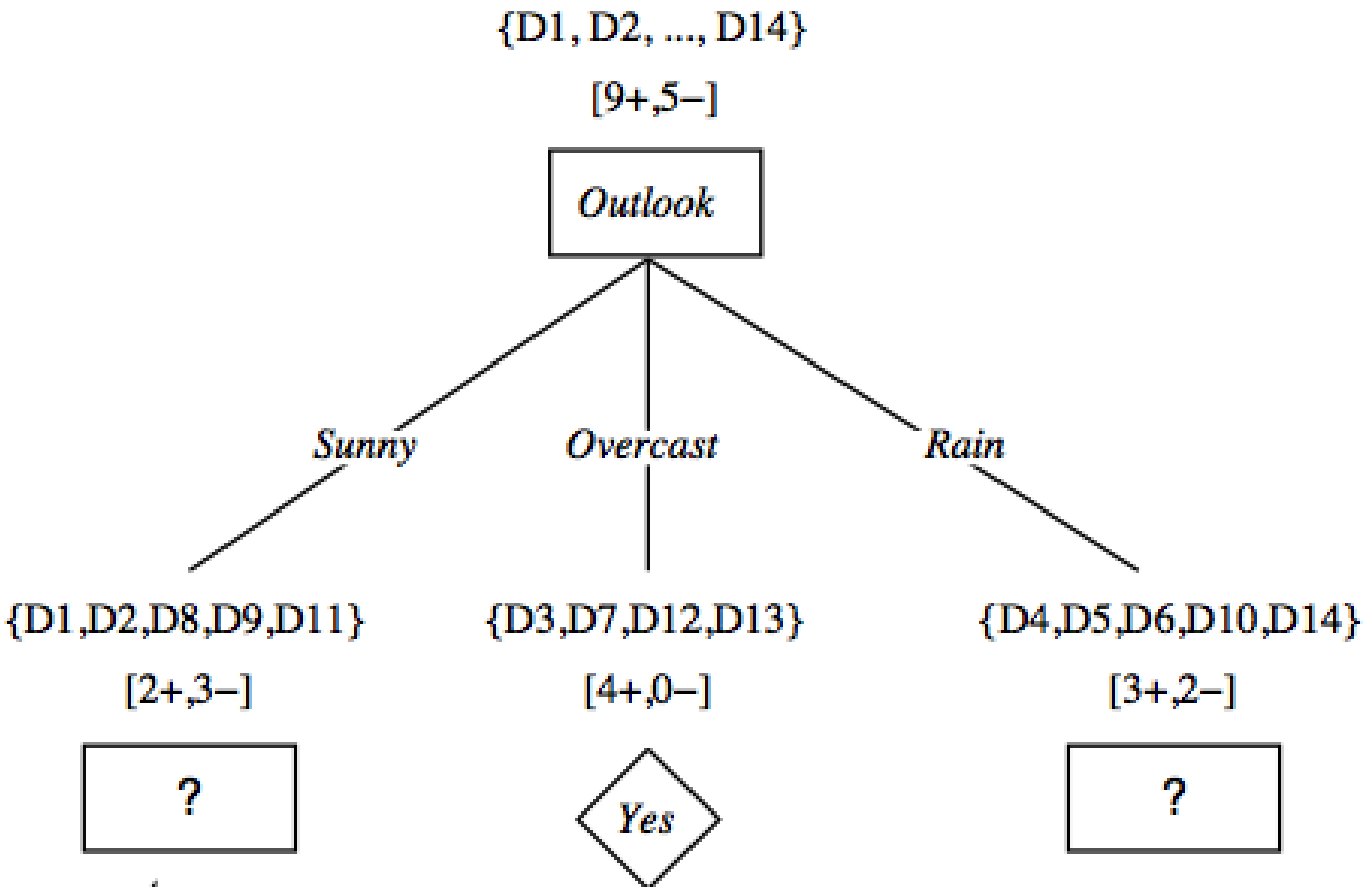
Which attribute is the best classifier?



First step: which attribute to test at the root?

- Which attribute should be tested at the root?
  - $Gain(S, Outlook) = 0.246$
  - $Gain(S, Humidity) = 0.151$
  - $Gain(S, Wind) = 0.084$
  - $Gain(S, Temperature) = 0.029$
- *Outlook* provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of *Outlook*
  - partition the training samples according to the value of *Outlook*

After first step



Second step

- Working on *Outlook=Sunny* node:

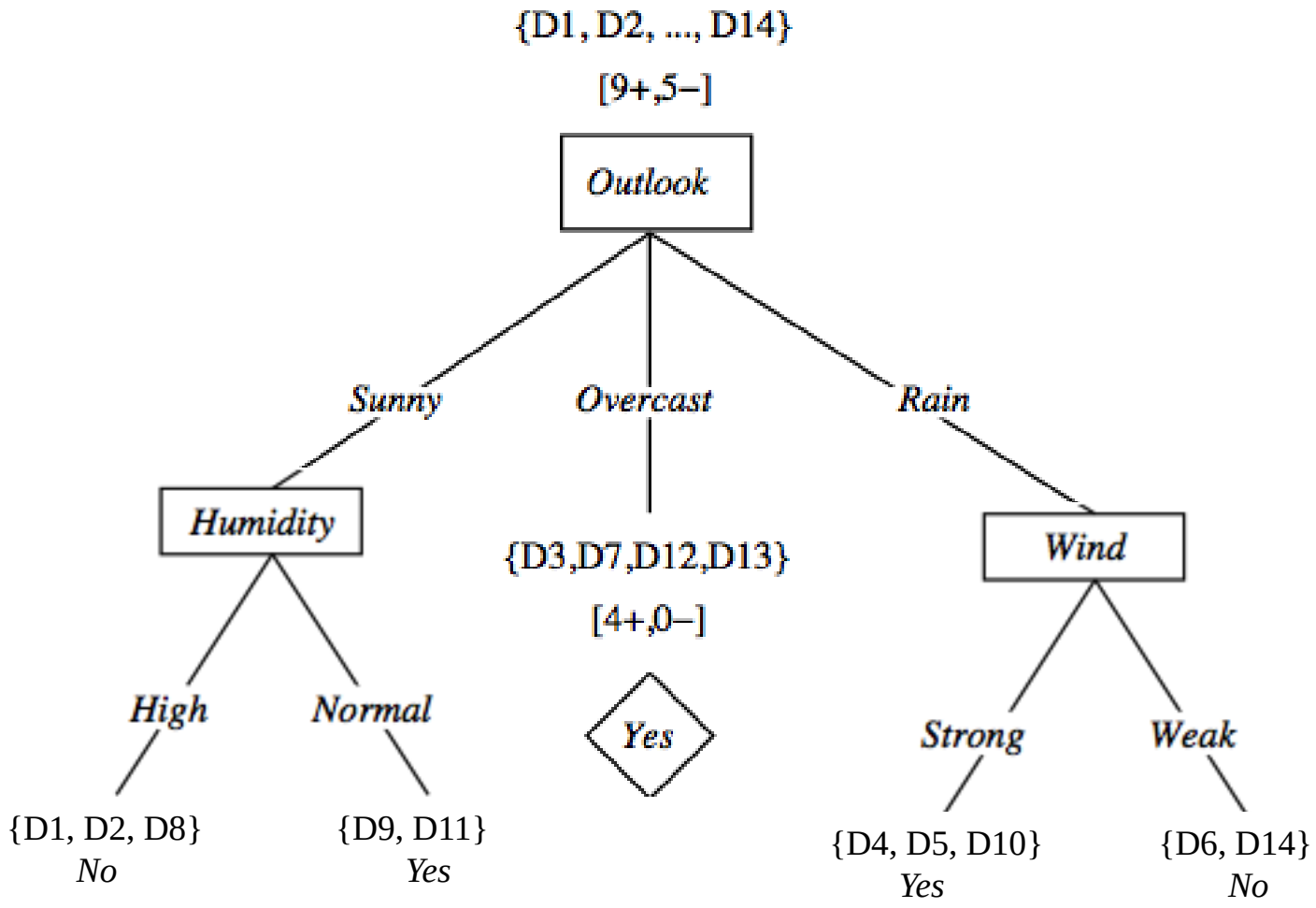
  $Gain(S_{Sunny}, Humidity) = 0.970 - 3/5 \times 0.0 - 2/5 \times 0.0 = 0.970$

  $Gain(S_{Sunny}, Wind) = 0.970 - 2/5 \times 1.0 - 3.5 \times 0.918 = 0\ .019$

  $Gain(S_{Sunny}, Temp.) = 0.970 - 2/5 \times 0.0 - 2/5 \times 1.0 - 1/5 \times 0.0 = 0.570$

- *Humidity* provides the best prediction for the target
- Lets grow the tree:
  - add to the tree a successor for each possible value of *Humidity*
  - partition the training samples according to the value of *Humidity*

Second and third steps



ID3: algorithm

ID3(*X, T, Attrs*) *X*: training examples:
T: target attribute (e.g. $PlayTennis$),
Attrs: other attributes, initially all attributes

Create *Root* node
If all X's are +, *return Root* with class +
If all X's are –, *return Root* with class –
If Attrs is empty *return Root* with class most common value of *T* in *X*
else
$A \leftarrow$ best attribute; decision attribute for Root $\leftarrow A$
For each possible value v_i of *A:*
- add a new branch below *Root,* for test $A = v_i$
- $X_i \leftarrow$ subset of *X* with $A = v_i$
- *If* X_i is empty *then* add a new leaf with class the most common value of *T* in *X*
else add the subtree generated by ID3(X_i, *T, Attrs* – {*A*})

return Root

Avoid overfitting in Decision Trees

- Two strategies:
  1. Stop growing the tree earlier the tree, before perfect classification
  2. Allow the tree to *overfit* the data, and then *post-prune* the tree
- Training and validation set
  - split the training in two parts (training and validation) and use validation to assess the utility of *post-pruning*
    - *Reduced error pruning*
    - *Rule post pruning*

Reduced-error pruning

- Each node is a candidate for pruning
- *Pruning* consists in removing a subtree rooted in a node: the node becomes a leaf and is assigned the most common classification
- Nodes are removed only if the resulting tree performs no worse on the validation set.
- Nodes are pruned iteratively: at each iteration the node whose removal most increases accuracy on the validation set is pruned.
- Pruning stops when no pruning increases accuracy

Rule post-pruning

1. Create the decision tree from the training set
2. Convert the tree into an equivalent set of rules
   - Each path corresponds to a rule
   - Each node along a path corresponds to a pre-condition
   - Each leaf classification to the post-condition
3. Prune (generalize) each rule by removing those preconditions whose removal improves accuracy …
   - … over validation set
4. Sort the rules in estimated order of accuracy, and consider them in sequence when classifying new instances

Why converting to rules?

- Each distinct path produces a different rule: a condition removal may be based on a local (contextual) criterion. Node pruning is global and affects all the rules
- Provides flexibility of not removing entire node
- In rule form, tests are not ordered and there is no book-keeping involved when conditions (nodes) are removed
- Converting to rules improves readability for humans

Dealing with continuous-valued attributes

- Given a continuous-valued attribute A, dynamically create a new attribute A_c

$$A_c = \text{True } if\ A < c, \text{ False } otherwise$$

- How to determine threshold value c ?
- Example. *Temperature* in the *PlayTennis* example
  - Sort the examples according to *Temperature*

Temperature	40	48		60	72	80		90
PlayTennis	No	No	*54*	Yes	Yes	Yes	*85*	No

  - Determine candidate thresholds by averaging consecutive values where there is a change in classification: (48+60)/2=54 and (80+90)/2=85

Problems with information gain

- Natural bias of information gain: it favors attributes with many possible values.
- Consider the attribute *Date* in the *PlayTennis* example.
  - *Date* would have the highest information gain since it perfectly separates the training data.
  - It would be selected at the root resulting in a very broad tree
  - Very good on the training, this tree would perform poorly in predicting unknown instances. Overfitting.
- The problem is that the partition is too specific, too many small classes are generated.
- We need to look at alternative measures …

An alternative measure: gain ratio

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- S_i are the sets obtained by partitioning on value i of A
- *SplitInformation* measures the entropy of S with respect to the values of A. The more uniformly dispersed the data the higher it is.

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

- *GainRatio* penalizes attributes that split examples in many small classes such as *Date.* Let $|S|=n$, *Date* splits examples in n classes
  - $SplitInformation(S, Date) = -[(1/n \log_2 1/n) + \ldots + (1/n \log_2 1/n)] = -\log_2 1/n = \log_2 n$
- Compare with A, which splits data in two even classes:
  - $SplitInformation(S, A) = -[(1/2 \log_2 1/2) + (1/2 \log_2 1/2)] = -[-1/2 - 1/2] = 1$

Handling missing values training data

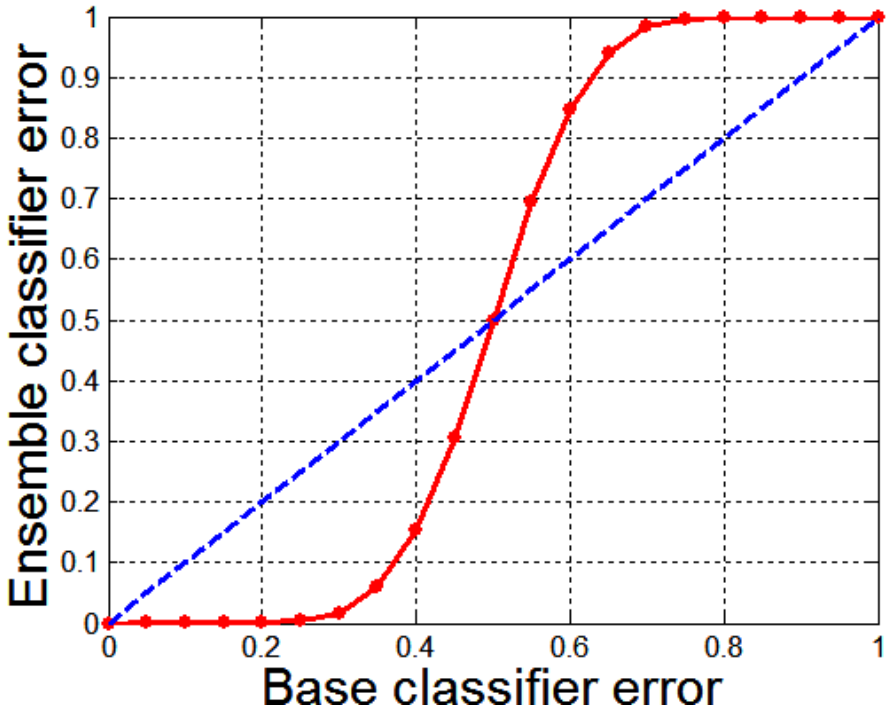
- How to cope with the problem that the value of some attribute may be missing?
- The strategy: use other examples to guess attribute
  1. Assign the value that is most common among the training examples at the node
  2. Assign a probability to each value, based on frequencies, and assign values to missing attribute, according to this probability distribution

Ensemble Methods

- **Ensemble methods** use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone
- Construct a set of classifiers from the training data
- Predict class label of test records by combining the predictions made by multiple classifiers
- Tend to reduce problems related to over-fitting of the training data.

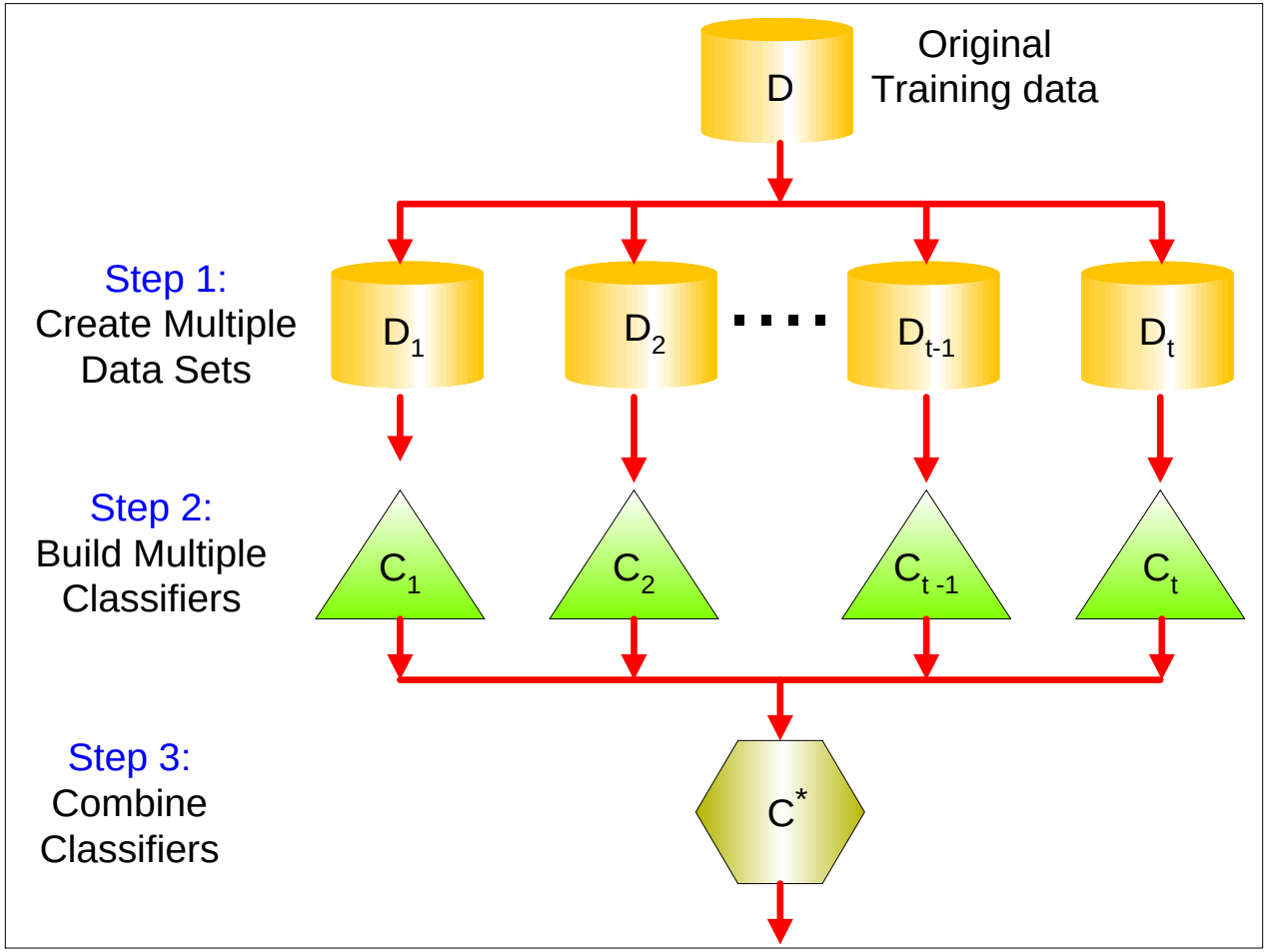
Why Ensemble Methods work?

- 25 base classifiers
- Each classifier has error rate, $\varepsilon = 0.35$
- If base classifiers are identical, then the ensemble will misclassify the same examples predicted incorrectly by the base classifiers depicted by dotted line
- Assume errors made by classifiers are uncorrelated
- ensemble makes a wrong prediction only if more than half of the base classifiers predict incorrectly
- Probability that the ensemble classifier makes a wrong prediction:



$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

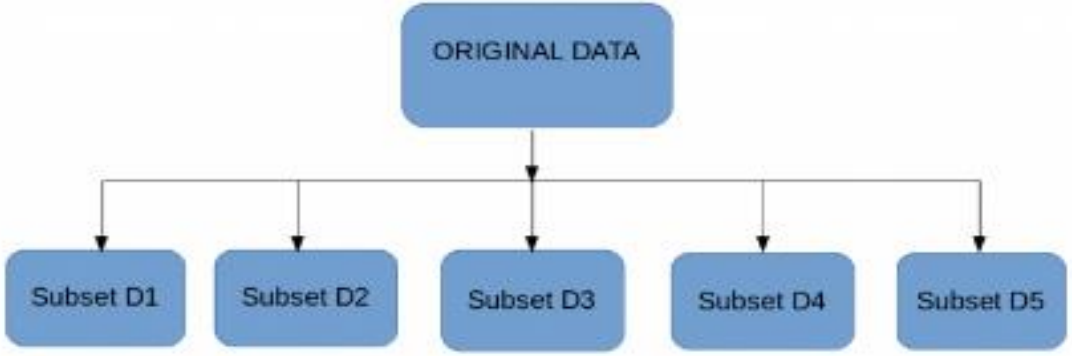
General Approach



Bagging (Bootstrap Aggregating)

- Technique uses these subsets (bags) to get a fair idea of the distribution (complete set).
- The size of subsets created for bagging may be less than the original set.
- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**.
- When you sample with replacement, items are independent. One item does not affect the outcome of the other. You have 1/7 chance of choosing the first item and a 1/7 chance of choosing the second item.
- If the two items are **dependent**, or linked to each other. When you choose the first item, you have a 1/7 probability of picking a item. Assuming you don't replace the item, you only have six items to pick from. That gives you a 1/6 chance of choosing a second item.

Bagging



- Multiple subsets are created from the original dataset, selecting observations with replacement.
- A base model (weak model) is created on each of these subsets.
- The models run in parallel and are independent of each other.
- The final predictions are determined by combining the predictions from all the models.

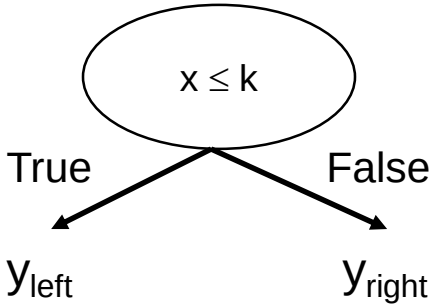
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

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

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 6:

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

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

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

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

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

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

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

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

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

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

- Summary of Training sets:

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Bagging Example

- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

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

Predicted Class

Bagging Algorithm

Algorithm 5.6 Bagging Algorithm

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size n, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \arg\max_y \sum_i \delta(C_i(x) = y)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise.$\}$

Random Forest

- Random Forest is another ensemble machine learning algorithm that follows the bagging technique.
- The base estimators in random forest are decision trees.
- Random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

Random Forest

- As in bagging, we build a number of decision trees on bootstrapped training samples each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
- Note that if m = p, then this is bagging.

Random Forest

- Random subsets are created from the original dataset (bootstrapping).
- At each node in the decision tree, only a random set of features are considered to decide the best split.
- A decision tree model is fitted on each of the subsets.
- The final prediction is calculated by averaging the predictions from all decision trees.

Random Forests Algorithm

- For b = 1 to B:
  - (a) Draw a bootstrap sample $Z*$ of size N from the training data.
  - (b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size nmin is reached.
    - i. Select m variables at random from the p variables.
    - ii. Pick the best variable/split-point among the m.
    - iii. Split the node into two daughter nodes. Output the ensemble of trees.
- To make a prediction at a new point x we do:
  - For regression: average the results
  - For classification: majority vote

Thank You