

Image Searching and Modelling

**Part IB Paper 8
Information Engineering Elective**

Lecture 3: Bayesian Learning and Nearest Neighbour

Zoubin Ghahramani

`zoubin@eng.cam.ac.uk`

**Department of Engineering
University of Cambridge**

Easter Term

Bayesian Learning and Binary Data

Imagine we arrive from overseas and start to learn about the weather in Cambridge.

Let $x_n = 1$ mean “rain” on day n , and $x_n = 0$ mean “no rain”.

The data set: $\mathcal{D} = \{1, 1, 1\}$ for first three days.

Let θ be the probability of rain (assume that rain is independent across days¹).

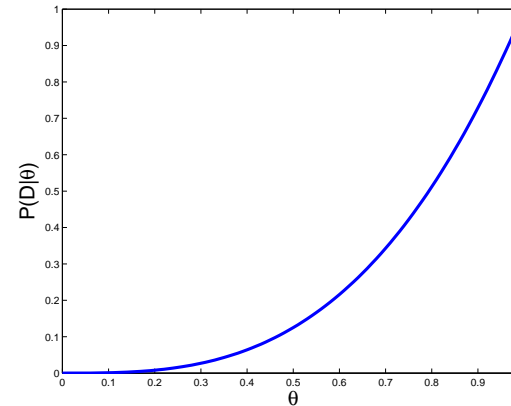
Q: What is the ML estimate of θ ? Is this reasonable?

Q: What does the likelihood look like?

¹clearly not realistic

Bayesian Learning and Binary Data

Q: What does the likelihood look like?



$$P(\mathcal{D}|\theta) = \theta^3$$

Assume a uniform prior on θ : $P(\theta) = 1$

Q: What is the posterior on θ , $P(\theta|\mathcal{D})$?

$$P(\theta|\mathcal{D}) = \theta^3 / (1/4) = 4\theta^3$$

Q: What is the expected value (mean) of θ given \mathcal{D} ?

$$E(\theta|\mathcal{D}) = \int_0^1 \theta P(\theta|\mathcal{D}) d\theta = 4 \int_0^1 \theta^4 d\theta = \frac{4}{5}$$

Q: What is the predicted probability of x given \mathcal{D} ?

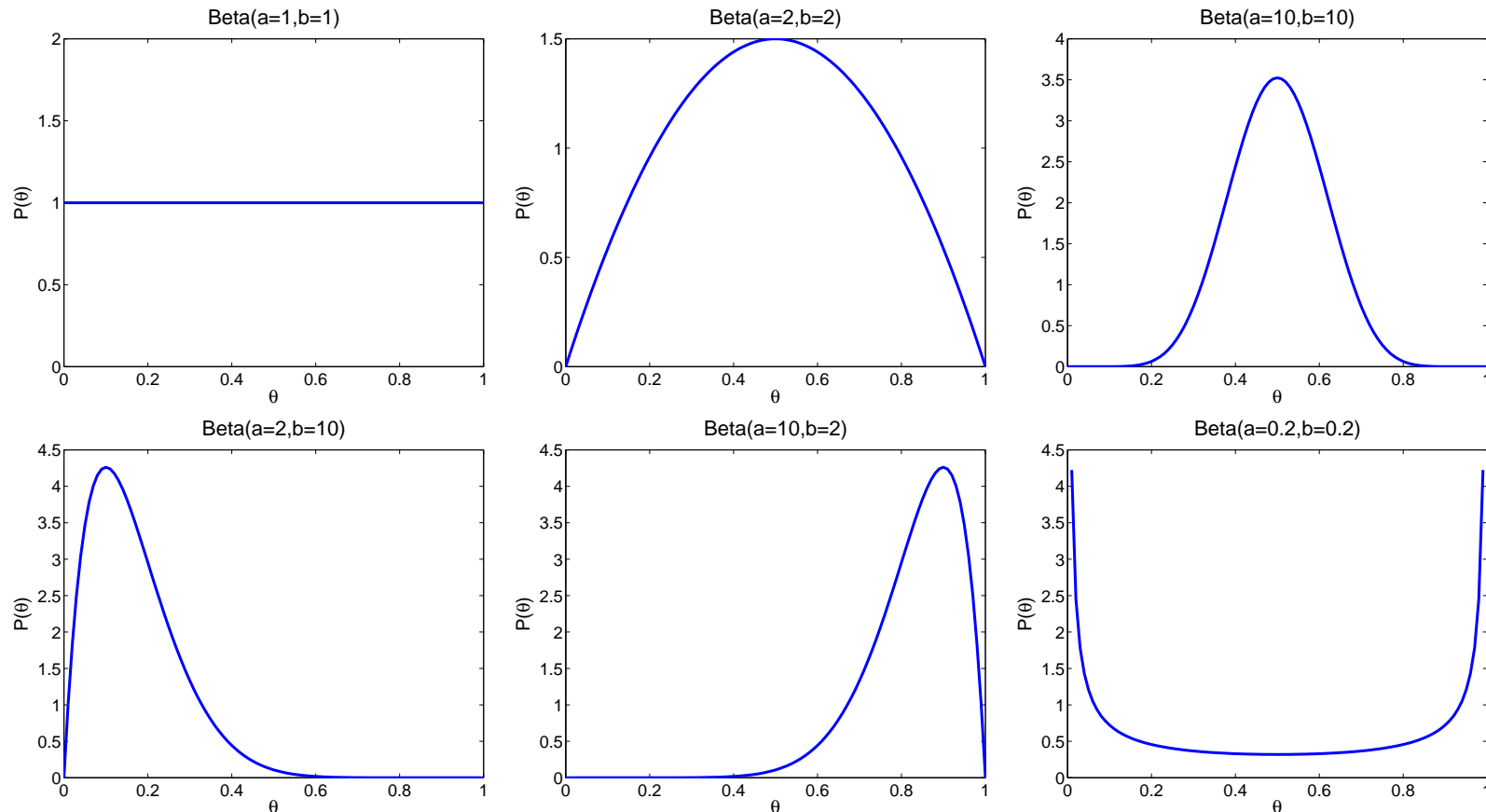
$$P(x = 1|\mathcal{D}) = \int P(x = 1|\theta) P(\theta|\mathcal{D}) d\theta = \int \theta P(\theta|\mathcal{D}) d\theta = \frac{4}{5}$$

The Beta distribution

The Beta distribution is: $P(\theta|a, b) = \frac{1}{c} \theta^{a-1} (1 - \theta)^{b-1}$

where c is a normalization constant making the distribution integrate to 1 over $[0, 1]$.²

Q: What does the Beta distribution look like?



$$c = \int_0^1 \theta^{a-1} (1 - \theta)^{b-1} d\theta$$

The Beta distribution

The Beta distribution is: $P(\theta|a, b) = \frac{1}{c}\theta^{a-1}(1 - \theta)^{b-1} = \text{Beta}(a, b)$

where c is a normalization constant making the distribution integrate to 1 over $[0, 1]$.

Compare the Beta distribution to the Bernoulli over x :

$$P(x|\theta) = \theta^x(1 - \theta)^{(1-x)}$$

Q: What happens if multiply a Beta prior and a Bernoulli likelihood?

$$\begin{aligned} P(\theta|x, a, b) &\propto P(\theta|a, b)P(x|\theta) \\ &= \frac{1}{c}\theta^{a-1}(1 - \theta)^{b-1}\theta^x(1 - \theta)^{(1-x)} \\ &= \frac{1}{c}\theta^{a+x-1}(1 - \theta)^{b+1-x-1} \propto \text{Beta}(a + x, b + 1 - x) \end{aligned}$$

Beta prior \times Bernoulli likelihood = Beta posterior³.

³This property is called *conjugacy*. Beta is the conjugate prior for the Bernoulli likelihood.

More on the Beta distribution

What is the normalizing constant?

$$P(\theta|a, b) = \frac{1}{c} \theta^{a-1} (1 - \theta)^{b-1} = \text{Beta}(a, b)$$

$$c = \int_0^1 \theta^{a-1} (1 - \theta)^{b-1} d\theta$$

$$c = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

where

$$\Gamma(a) = \int_0^{\infty} t^{a-1} e^{-t} dt$$

The Gamma function is a generalization of the factorial function. For integer a :

$$\Gamma(a) = (a - 1)!$$

[You will not be examined on the definition of the Gamma function]

Nearest Neighbor

Nearest Neighbor

Given a set of points $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ and some other point \mathbf{x} , we want to find the *nearest neighbor* of \mathbf{x} in \mathcal{D} .

$$n^* = \operatorname{argmin}_{n \in \{1, \dots, N\}} d(\mathbf{x}, \mathbf{x}_n)$$

where $d(\mathbf{x}, \mathbf{x}')$ is the distance between \mathbf{x} and \mathbf{x}' .

For example:

$$d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|$$

Simplest algorithm:

Initialize: $n^* = 1$

For $n = 1$ to N

If $d(\mathbf{x}, \mathbf{x}_n) < d(\mathbf{x}, \mathbf{x}_{n^*})$ then $n^* \leftarrow n$ endif;

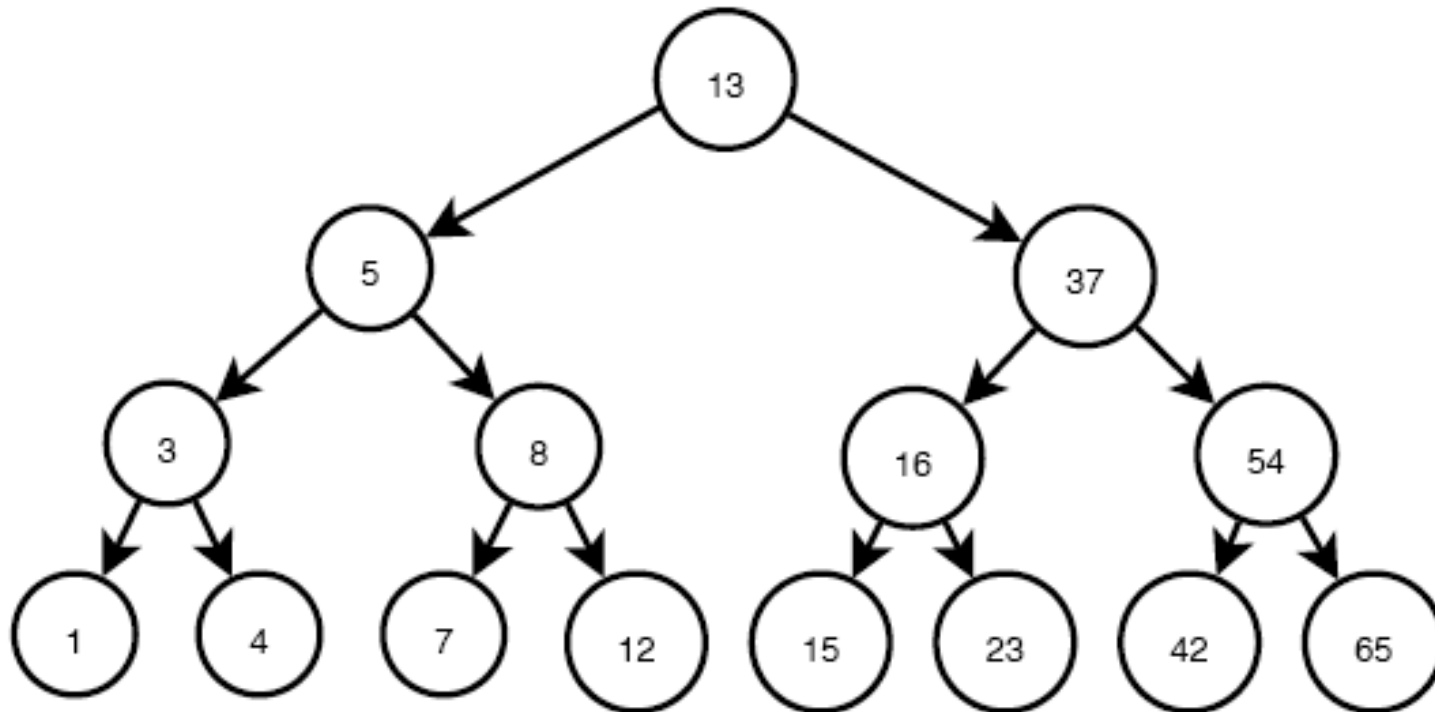
Endfor

This algorithm is $\mathcal{O}(N)$.

Q: Can we do better than this?

Using good data structures

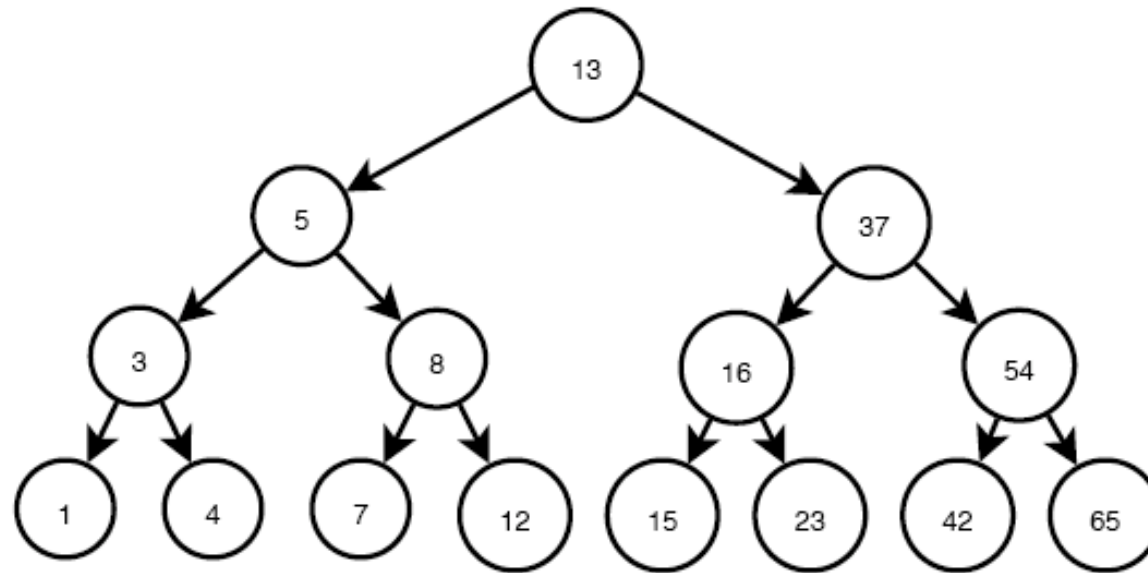
Search Trees!



The structure of a node is quite simple:

```
structure Node
{
    scalar value;
    Node leftLeaf;
    Node rightLeaf;
}
```

Search Trees



The algorithm to search the tree is a **while** loop:

```
procedure search(Node root, scalar query) returns boolean
{
  Node current = root;
  while(current is something)
  {
    if(current.value == query)
      return true;
    if(query < current.value)
      current = current.leftLeaf;
    else current = current.rightLeaf;
  }
  return false;
}
```

The procedure to add a new node to the tree is recursive:

```
procedure add(Node current, scalar value) returns Node
{
  if(current is nothing)
    return new Node(value, nothing, nothing);
  if(value == current.value)
    return current;
  if(value < current.value)
    current.leftLeaf =
      add(current.leftLeaf, value);
  else current.rightLeaf =
    add(current.rightLeaf, value);
  return current;
}
```