NETWORK INFERENCE

Duncan Gillies
Imperial College, London

Lecture 1

Graph Models

Communication Networks

Networks have always been with us, and have important social and strategic functions.

AQUILEIA
GALLIA
CISA
VERONA
PLACENTIA
PARMA
BONONIA
LIGURIA
ARIMINUM
FANUM
GENUA
SAVO
ANCONA
CASTRUM TRUENTUM
ASCULUM
ATERNUM
LIBURNIA
ILLYRICUM
TYRRHENUM
TERRACINA
BRUNDISIUM
TARENTUM

The Romans built the first European road network.

It brought civilisation and prosperity to many countries - and huge wealth to Rome.

Communication equated with transport in those days!

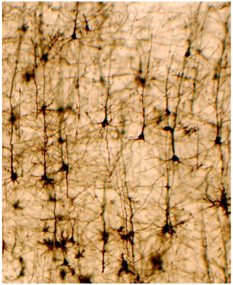
Communication Networks

Communication networks in their own right have evolved dramatically over the last 250 years:

- Semaphore signalling towers (London-Portsmouth in 15 mins)
- Wheatstone electrical telegraph 1830.
- Telephone 1876
- The radio telegraph (Marconi) 1909
- Satellite communication (Sputnik 1957)
- ARPA Net 1963
- Fibre optic nets 1970s

Communication Networks

There are also biological communications networks.

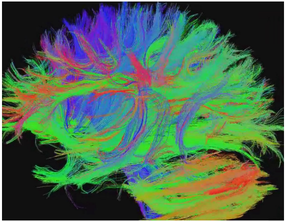


In the ninteenth century scientists learnt how to stain neurones to make them visible under the microscope.

They discovered that the brain was a highly complex network

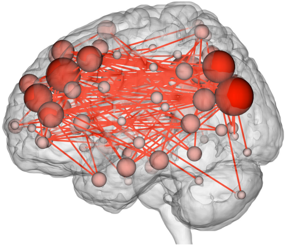
It looks as if networks are an important component of decision making.

Communication Networks



Diffusion tensor medical imaging has been able to identify connection pathways within the brain:

Functional medical imaging has been able to map the areas of the brain invloved in thought patterns:



and there are genetic networks, vascular networks, endocrine networks, social networks, etc. etc.

Why Study Networks

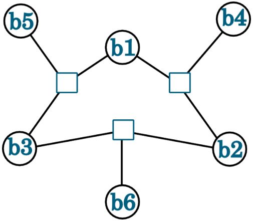
We can use the data we gain from studying networks to:

- Try to understand how they function, for example by investigating spiking neural rate models.
- Create biologically inspired machines, for example Deep Neural Networks
- Use network inference to solve engineering problems.

We will start by looking at two examples network inference.

The Tanner graph

- The Tanner graph is a graphical model that was designed for error recovery in parity checking.
- It is a bi-partite graph - ie it has two node types and each node connects only to its opposite type.



- The circles represent bits and the squares represent parity checks.
- The squares evaluate to 1 if the sum of the bits is even.

Parity checking as an inference problem

Suppose the bits are transmitted down a noisy channel and let P_f be the probability that a bit is flipped during transmission.

- Let Y_i be the measured (received) bit values.
- Let X_i be the true (transmitted) bit values which we want to recover

then:

$$\begin{aligned} P(X_i|Y_i) &= \quad 1 - P_f \quad \text{if } X_i = Y_i \\ &= \quad P_f \quad \text{otherwise} \end{aligned}$$

Parity checking as an inference problem

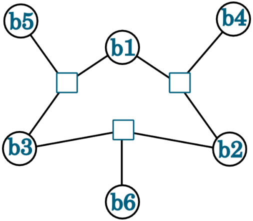
So given a possible bit string $(X_1, X_2, X_3, ...X_n)$ we can calculate a probability of it being correct using:

$$P(X_1, X_2, X_3, ...X_n) = \prod_{j=1}^{n} P(X_j | Y_j)$$

If we did not have any parity constraints then the most probable bit string is the one for which $X_i = Y_i$ for all the bits - ie the string we received.

Parity checking as an inference problem

Now suppose that bits 1,2 and 3 are data bits and 4, 5 and 6 are parity bits.



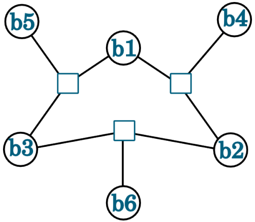
The parity bits are set so that sum of the three bits in each group is even. This can be expressed as a constraint function for each square:

$$\Psi(X_1, X_3, X_5) = 1 - (X_1 + X_3 + X_5) mod 2$$

Parity checking as an inference problem

We want to ensure that if there is a parity check failure the we reject the received bit string completely.

To do this we write the joint probability distribution of a bit string as:



$$P(X_1,X_2\cdots X_6)=\Psi(X_1,X_3,X_5)\Psi(X_1,X_3,X_5)\Psi(X_1,X_3,X_5)\prod_{j=1}^{n}P(X_j|Y_j)$$

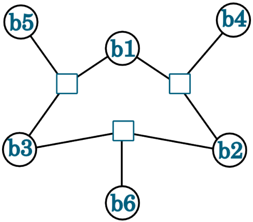
In the event that the received bit string fails a parity check the next most probable bit string is selected.

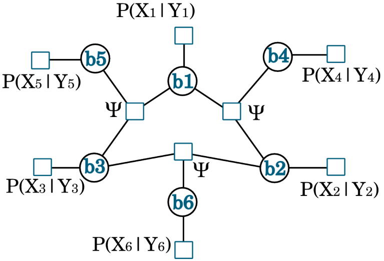
The Tanner graph as a factor graph

The Tanner graph is a factorisation of a probability distribution, in our example it has 9 factors:

$$P(X_1, X_2 \cdots X_6) = \Psi(X_1, X_3, X_5)\Psi(X_1, X_3, X_5)\Psi(X_1, X_3, X_5)\prod_{j=1}^{n} P(X_j|Y_j)$$

This is made clearer by including the individual bit probabilities as boxes (factors) in the graph:





Factor Graphs

Factorisation of probability distributions is a fundamental idea in probabilistic inference, and we will return to it later.

The factor graph is a the most general graphical way to express probabilistic inference problems.

Circles represent data and squares represent factors which multiply together to form the joint probability of the data

$$P(X_1, X_2, X_3, \dots X_n) = \prod_{j=1}^{m} f_j$$

Where each f_j is a function of a subset of the variables.

Markov Random Fields

Markov Random Fields are graphical models in which each node depends only on its immediate neighbours.

They are expressed as a factorisation into subsets V_j each with a potential function:

$$P(X_1, X_2, X_3, ... X_n) = \frac{1}{Z} \prod_{j=1}^{m} \Psi_j(V_j)$$

Where Z is a normalisation constant:

$$Z = \sum_{X} \prod_{j=1}^{m} \Psi_j(V_j)$$

Markov Random Fields

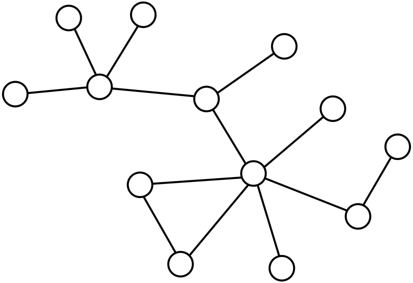
The normalisation constant Z gives us flexibility in the definition the factors in a Markov Random Field.

$$Z = \sum_{X} \prod_{j=1}^{m} \Psi_j(V_j)$$

However this comes at a computational cost, as for large inference problems Z is difficult (possibly infeasible) to compute.

Markov Random Fields

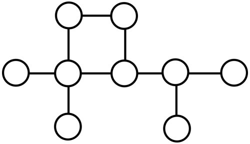
Each node in a MRF can correspond to a single variable or a set of variables, and each arc implies a relationship between the variables it joins.



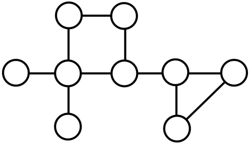
For example an arc might indicate that two variables are strongly correlated.

Pairwise Markov Random Fields

A pairwise Markov random field has all its variables joined in cliques of size 2:

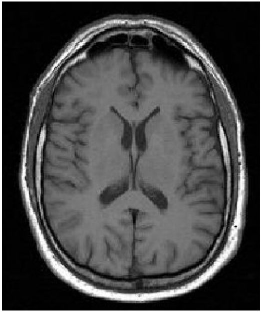


but not:



The corresponding factor graph has factors that join at most two variables.

Example - image segmentation and restoration



Pairwise Markov Random Fields have been used successfully in image processing for medical diagnosis.

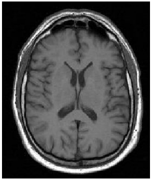
Segmentation: determine which class pixels belong to:

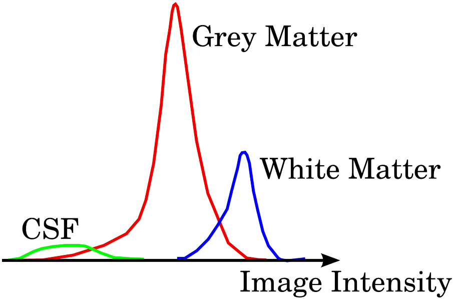
- GM: Grey Matter (cortical neurones)
- WM: White Matter (connective tissue)
- CSF: Cerebral Spinal Fluid

Restoration: correct pixels that are wrong due to imaging errors and artefacts.

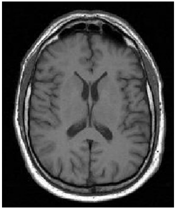
Example - image segmentation and restoration

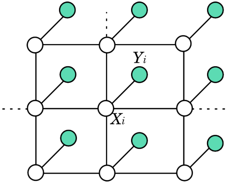
In the image segmentation problem each pixel is a discrete variable which can have one of three possible values or states. The intensity ranges of these states in a medical image will overlap, meaning that thresholding the image does not give an accurate segmentation.





Example - image segmentation and restoration





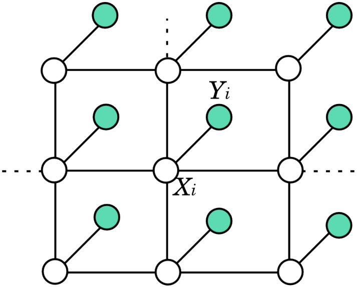
Each pixel in the image is a variable in the inference problem. The filled circles represent actual pixel values and are labelled Yi.

The model (segments or restored image values) is represented by the empty circles Xi.

The Xi values are calculated using the pixel measurements and the neighbour's messages.

Factor graph of the Pairwise MRF

The pairwise Markov random field, being just a factorisation of a probability distribution can be represented as a factor graph.



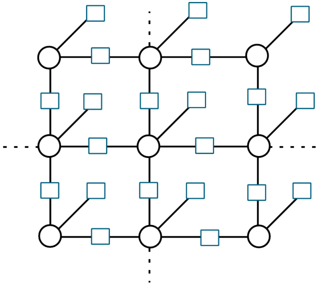


Image Segmentation as Probabilistic Inference

We need to define two types of factor which are commonly called compatability functions.

$\Phi(X_i, Y_i)$ - relates the observed and hidden values. It is rather like a conditional probability $P(X_i|Y_i)$. It expresses the probability of the pixel belonging to a particular class (WM, GM, CSF) given the measured pixel value Yi.

$\Psi(X_i, X_j)$ - expresses the compatibility between adjacent pixels. Any pixels not connected will have a $\Psi(X_i, X_j)$ value of 1 expressing no information. For connected pixels this compatibility function is like a joint probability of the adjacent states being neighbours.

Image segmentation as Probabilistic Inference

Given an image $Y = (Y_1, Y_2, ..Y_n)$
And a segmentation $X = (X_1, X_2, ..X_n)$
We can define a joint probability distribution:

$$P(X,Y) = \frac{1}{Z}\prod_i \Phi_j(X_i, Y_i)\prod_{i,j}\Psi_j(X_i, X_j)$$

and find the values of X_i that give the maximum probability (ie the most likely segmentation)

Oh Dear!! We are in trouble! ???

The high cost of computing Z makes this direct approach computationally infeasible.

Belief Propagation

Belief propagation overcomes the computational difficulties of using a global joint probability distribution by making local computations.

In the case of image segmentation each pixel could be in one of three possible states: “grey matter (GM)”, “white matter (WM)” and “fluid (CSF)”. Its belief is just the probabilty distribution over theses states.

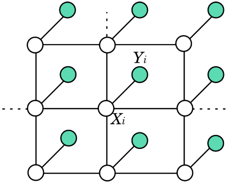
In belief propagation each variable will send a message to each of it’s neighbours and its neighbours will then update their belief.

The belief in a variable is just its posterior probability distribution.

Belief propagation in MRFs

We write the belief in one state of a variable (eg GM, WM or CSF) as:

$$b(X_i(s)) = \frac{1}{Z}\Phi(X_i(s), Y_i) \prod_{k\in N(i)} m_k(X_i(s))$$



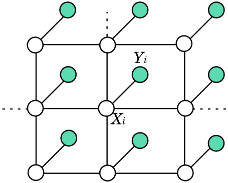
Where:

- $X_i(s)$ means state s of node X_i
- m_k means a message (or evidence) from neighbour k
- $N(i)$ is the set of neighbours of i

Belief propagation in MRFs

It is also convenient to define:

$$b_{\backslash j}(X_i(s)) = \frac{1}{Z}\Phi(X_i(s), Y_i) \prod_{k \in N(i)\backslash j} m_k(X_i(s))$$



Where: $\backslash j$ means excluding neighbour j

If node i is going to send a message to node j it must exclude any information it got from j

Belief propagation in MRFs

Finally we can define the message that node *i* will send to node *j*.

$$\mathbf{m_i} = \mathbf{b_{\backslash j}}(\mathbf{X_i})\Psi(X_i, X_j)$$

Where:

- $\mathbf{m_i}$ is a vector message from X_i for the states of X_j
- $\mathbf{b_{\backslash j}}(\mathbf{X_i})$ is a vector of the beliefs in the states of Xi excluding the evidence from X_j
- $\Psi(X_i, X_j)$ is the compatibility matrix.

Terminating Belief Propagation

In one epoch of belief propagation each node may send a message to each of it's neighbours. But how many epochs do we need?

There is no definitive answer to this question as the process is highly data dependent.

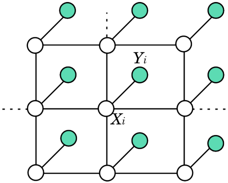
One possibility is to record the total change in belief of each pixel in each epoch, and terminate when this change reaches a minimum.

This is the problem of Loopy Belief Propagation and is common in network inference.

Belief propagation in MRFs

Notice that part of the belief of each variable $\Phi(X_i(s), Y_i)$ is fixed for all epochs.

$$b(X_i(s)) = \frac{1}{Z}\Phi(X_i(s), Y_i) \prod_{k \in N(i)} m_k(X_i(s))$$



This will hlep to stabilise the process leading to convergence.

Computational characteristics of Graphical Models

Without using a graphical model a probabilistic inference about a variable may be expressed as a sum over the entire joint distribution of the variables

But this is clearly infeasible for even moderate sized problems.

However the graphical model has given us a feasible approach to making inferences by expressing the topology of the variables.

Extracting Networks from Data

In both the previous examples the network was designed to match the inference problem.

We used the fact that there is more likely to be a relationship between adjacent pixels, than between distant pixels.

Another interesting application of graph models is in discovering relationships between variables in a data set.

The Affinity Matrix

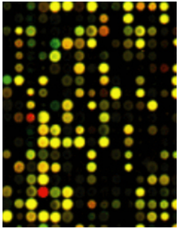
Given that we have a data set in which there are n variables and N data points (samples) we can construct two possible affinity matrices:

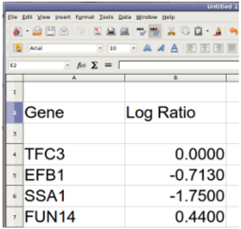
- The variable affinity matrix which has dimension $n \times n$ and each entry expresses the affinity (or similarlty) between a pair of variables.

- The sample affinity matrix which has dimension $N \times N$ and each entry expresses the affinity (or similarlty) between a pair of samples.

An example: Gene Regulatory Networks

Each gene is a spot on a glass slide whose colour indicates the difference between a normal and cancerous sample. Each gene has a single measured number ranging (potentially) from $-\infty$ (cancerous sample only) to $+\infty$ (normal sample only).



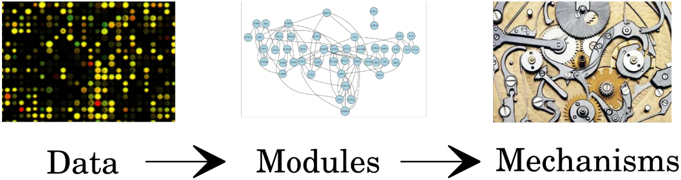


Gene	Log Ratio
TFC3	0.0000
EFB1	-0.7130
SSA1	-1.7500
FUN14	0.4400

There are typically 20000 genes in an experiment.

Modelling Regulatory Networks

The most widely adopted approach is a hierarchical one:



Data → Modules → Mechanisms

However there are many problems

- Experimental Error
- Time Resolution
- Overlapping patterns
- Large numbers of genes, but few experimental runs

The Microarry-Microarray Affinity Matrix

One way to build an affinity matrix is to define a distance: $D(\mu_i, \mu_j)$.

For example we could define the distance between a pair of microarrays (patient cases) in the same study as the Euclidian distance between the gene values.

This could be converted to an affinity value by using a Gaussian $A(\mu_i, \mu_j) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(\frac{D(\mu_i,\mu_j)}{2\sigma^2})^2}$

The Gene-Gene Affinity Matrix

Another method is to used covariance directly. Let an $N \times n$ data (or design) matrix $\boldsymbol{D}$ be composed of N sample points (microarrays) with n variables (genes). Each row is one sample of our data set.

$$\boldsymbol{D} = \begin{bmatrix} 150 & 152 & \cdots & 254 & 255 & \cdots & 252 \\ 131 & 133 & \cdots & 221 & 223 & \cdots & 241 \\ . & . & & . & . & & . \\ . & . & & . & . & & . \\ 144 & 171 & \cdots & 244 & 245 & \cdots & 223 \end{bmatrix}_{N \times n}$$

Suppose the mean of the columns of $\boldsymbol{D}$ (the average image) is:

$$[\ 120 \quad 140 \quad \cdots \quad 230 \quad 230 \quad \cdots \quad 240]$$

Mean Centring the data

The origin is moved to the mean of the data by subtracting the column average from each row.

$$
\boldsymbol{D} = \begin{bmatrix} 150 - 120 & \cdots & 254 - 230 & \cdots & 252 - 240 \\ 131 - 120 & \cdots & 221 - 230 & \cdots & 241 - 240 \\ . & & . & & . \\ . & & . & & . \\ 144 - 120 & \cdots & 244 - 230 & \cdots & 223 - 240 \end{bmatrix} N \times n
$$

This creates the mean centred data matrix:

$$
\boldsymbol{U} = \begin{bmatrix} 30 & 12 & \cdots & 24 & 25 & \cdots & 12 \\ 11 & -7 & \cdots & -9 & -7 & \cdots & 1 \\ . & . & & . & . & & . \\ 24 & 31 & \cdots & 14 & 15 & \cdots & -17 \end{bmatrix} N \times n
$$

Calculating the covariance matrix

The covariance matrix $\boldsymbol{\Sigma}$ can be calculated easily from the mean centered data matrix:

$$\Sigma = \boldsymbol{U}^T \boldsymbol{U} / (N - 1)$$

N is the number of data points (microarrays) and the covariance matrix has dimension $n \times n$.

Notice that we can make a different estimate of the microarray-microarray affinity matrix by computing:

$$\Sigma = \boldsymbol{U} \boldsymbol{U}^T / (n - 1)$$

Getting a Network from an Affinity Matrix

Having created an affinity matrix we can extract a network in different ways.

In every case we will probably want to apply a threshold below which we consider the gene pairs to be unrelated.

The threshold will determine the number of arcs in the graph.

We may wish to combine some prior knowledge (in this case known gene interactions) with the experimental results.

The Maximally Weighted Spanning Tree

For some applications we may want to extract just a spanning tree. To do this we can use the maximally weighted spanning tree algorithm:

- From the matrix extract a list of all gene pairs and their affinities.
- Sort the list in descending order of affinity
- Add arcs to the graph in the order they appear on the sorted list, but discarding arcs which if added form a loop.

The result is the tree that expresses the strongest dependency between the variables.

Dependency between Discrete Variables

We used discrete variables in the image segmentation example - each variable (pixel) could take one of three values or states: GM, WM or CSF, and belief propagation calculates a probability distribution over the states.

We can use conditional probability to determine degree of dependency of a pair of discrete variables. If they are independent:

$$P(D\&S) = P(D) \times P(S)$$

but if they are dependent in any way:

$$P(D\&S) = P(D) \times P(S|D)$$

Comparing $P(S\&D)$ with $P(S) \times P(D)$ is the basis of these dependency measures.

Dependency Measurement using an L1 metric

A joint probability, such as $P(B\&A)$, may be smaller or larger than the product of the individual probabilities ($P(B) \times P(B)$). For the simplest dependency measure we take the magnitude of the difference.

$$Dep(A,B) = |P(A\&B) - P(A)P(B)|$$

Summing over the joint states

In order to compute:

$$Dep(A,B) = |P(A\&B) - P(A)P(B)|$$

we need to sum over the joint states of the variables:

$$Dep(A,B) = \textstyle\sum_{A\times B} |P(a_i\&b_j) - P(a_i)P(b_j)|$$

This means we need to compute the distributions $P(A\&B)$, $P(A)$ and $P(B)$ from the data set.

Computing $P(A\&B)$

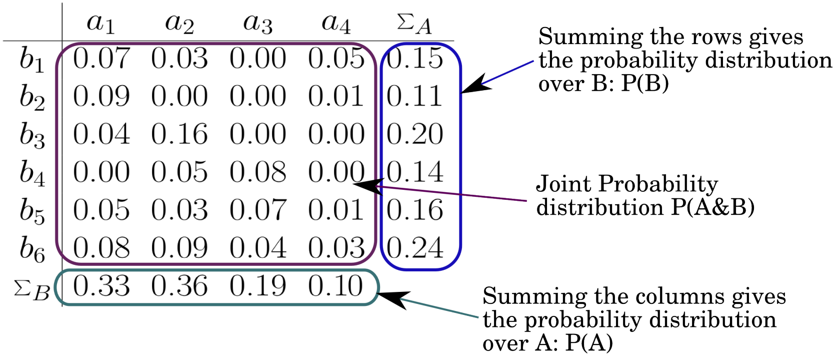
From the data set we first find the co-occurence matrix:

$$
\begin{array}{cc|cccc}
 & & & \text{A} & & \\
 & & a_1 & a_2 & a_3 & a_4 \\
\hline
 & b_1 & 5 & 2 & 0 & 4 \\
 & b_2 & 7 & 0 & 0 & 1 \\
\text{B} & b_3 & 3 & 12 & 0 & 0 \\
 & b_4 & 0 & 4 & 6 & 0 \\
 & b_5 & 4 & 2 & 5 & 1 \\
 & b_6 & 6 & 7 & 3 & 2
\end{array}
$$

$[a_2, b_3]$ occurs 12 times in the data set

We convert this to probabilities by dividing by the number of data points. (74 in this example)

Computing $P(A)$ and $P(B)$ by marginalisation



	a_1	a_2	a_3	a_4	Σ_A
b_1	0.07	0.03	0.00	0.05	0.15
b_2	0.09	0.00	0.00	0.01	0.11
b_3	0.04	0.16	0.00	0.00	0.20
b_4	0.00	0.05	0.08	0.00	0.14
b_5	0.05	0.03	0.07	0.01	0.16
b_6	0.08	0.09	0.04	0.03	0.24
Σ_B	0.33	0.36	0.19	0.10	

Summing the rows gives the probability distribution over B: P(B)

Joint Probability distribution P(A&B)

Summing the columns gives the probability distribution over A: P(A)

Other measures of dependency

- The measure we have used so far is an unweighted L1 metric.
- Its characteristic is that as the probabilities become small they contribute less to the dependency.
- This effect reflects the fact that we have little information on rare events

The Weighted L1 Metric

Another metric is formed by weighting the difference in magnitude by the joint probability:

$$Dep(A,B) = \sum_{A\times B} P(a_i \& b_j) \times |P(a_i \& b_j) - P(a_i)P(b_j)|$$

The effect is to further reduce the contribution to the dependency measure where probabilities are low.

The L2 Metric

L2 metrics use the squared differences:

$$Dep(A,B) = \sum_{A \times B} (P(a_i \& b_j) - P(a_i)P(b_j))^2$$

There is a weighted form:

$$Dep(A,B) = \sum_{A \times B} P(a_i \& b_j) \times (P(a_i \& b_j) - P(a_i)P(b_j))^2$$

Mutual Entropy

The most widely used measure in comparing probability distributions is Mutual Entropy. It is also called Mutual Information or the Kullback-Liebler divergence.

$$Dep(A,B) = \sum_{A\times B} P(a_i \& b_j) log_2((P(a_i \& b_j)/(P(a_i)P(b_j)))$$

- It is zero when two variables are completely independent
- It is positive and increasing with dependency when applied to probability distributions
- It is independent of the actual value of the probability