Machine Learning

4771

Instructors:
Adrian Weller and Ilia Vovsha

Lecture 16

- Update on course so far: instructors, TAs, midterm, HW3
- Review Clustering, K-Means (15:13-21)
- Mixture Models and Hidden Variables
- Expectation Maximization for Gaussian Mixtures
- Entropy, KL Divergence, Jensen's Inequality

Example: Vector Quantization

- Use K-means for lossy data compression

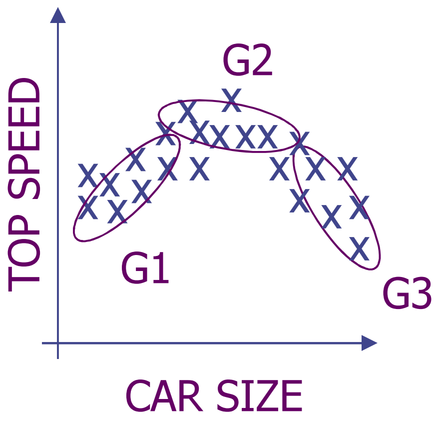
- Each pixel is a point in 3D (R,G,B) space. Instead need only store ?

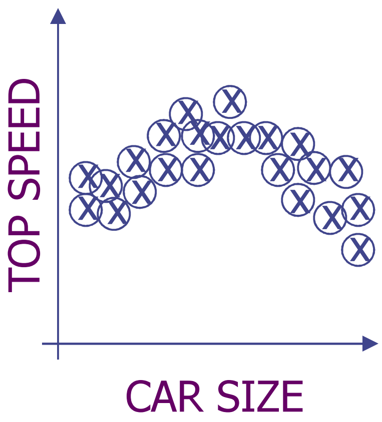
Mixtures for Flexibility

- With mixtures (e.g. mixtures of Gaussians) we can handle complicated distributions (e.g. multi-bump, nonlinear).

  subpopulations: G1=compact car
  G2=mid-size car
  G3=cadillac

- In fact, if we have enough Gaussians (maybe infinite) we can approximate any distribution...

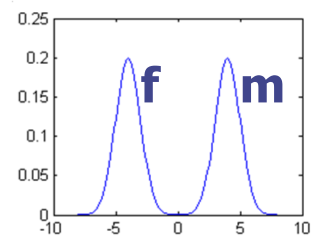




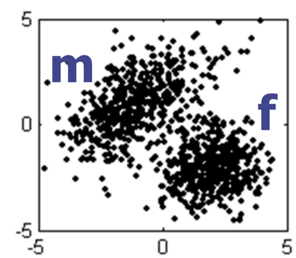
Mixtures as Hidden Variables

- Consider a dataset with K subpopulations but don't know which subpopulation each point belongs to

  e.g. consider height of adult people, we see K=2 subpopulations: males & females



  e.g. looking at weight and height of people we see K=2 subpopulations: males & females

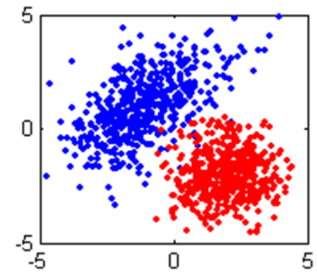


- Because of the 'hidden' variable (z can be 1 or 2), these distributions are not Gaussians but Mixture of Gaussians

$$p\left(\vec{x}\right) = \sum_z p(\vec{x}, z) = \sum_z p\left(z\right) p\left(\vec{x} \mid z\right) = \sum_z \pi_z N\left(\vec{x} \mid \vec{\mu}_z, \Sigma_z\right)$$

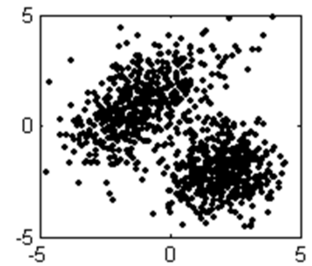
$$= \sum_{k=1}^{K} \pi_k \frac{1}{\left(2\pi\right)^{D/2} \sqrt{\left|\Sigma_k\right|}} \exp\left(-\frac{1}{2}\left(\vec{x} - \vec{\mu}_k\right)^T \Sigma_k^{-1} \left(\vec{x} - \vec{\mu}_k\right)\right)$$

Hidden / Unlabeled = Clustering



- Recall classification problem:
  maximize the log-likelihood:

$$l(\pi,\mu,\Sigma) = \sum_{n=1}^{N} \log p\left(\vec{x}_n, z_n \mid \pi,\mu,\Sigma\right) = \sum_{n=1}^{N} \log \pi_k N\left(\vec{x}_n \mid \vec{\mu}_k, \Sigma_k\right)$$



- If we don't know the class,
  marginalize over hidden variable

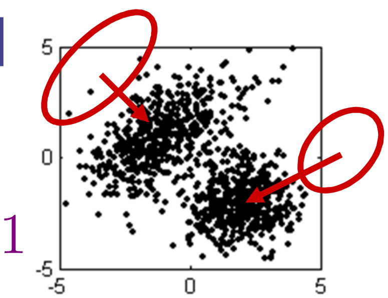
  maximize the log-likelihood with
  unlabeled data:

$$l = \sum_{n=1}^{N} \log p\left(\vec{x}_n \mid \pi,\mu,\Sigma\right) = \sum_{n=1}^{N} \log \sum_{z=1}^{K} p\left(\vec{x}_n, z \mid \pi,\mu,\Sigma\right) = \sum_{n=1}^{N} \log\left(\pi_1 N\left(\vec{x}_n \mid \vec{\mu}_1, \Sigma_1\right) + \ldots + \pi_K N\left(\vec{x}_n \mid \vec{\mu}_K, \Sigma_K\right)\right)$$

- Instead of classification, we now have a clustering problem

Mixture of Gaussians

- Represent each hidden z integer (1 to K) coded as a hidden binary indicator vector z



$$\vec{z} \in \mathbb{B}^K, \sum_{k=1}^K \vec{z}(k) = 1 \; or \; \vec{z} \in \{\vec{\delta}_1, \ldots, \vec{\delta}_K\} \; where \; \vec{\delta}_k(k) = 1$$

- Each likelihood requires summing over all possible z

$$p(\vec{x} \mid \theta) = \sum_z p(\vec{z} \mid \theta) p(\vec{x} \mid \vec{z}, \theta) = \sum_{k=1}^K p(\vec{z} = \vec{\delta}_k \mid \theta) p(\vec{x} \mid \vec{z} = \vec{\delta}_k, \theta)$$

mixing proportions (prior) = $\pi_k = p(\vec{z} = \vec{\delta}_k \mid \theta)$

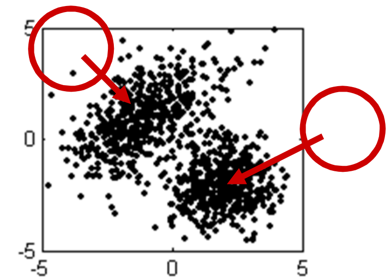
mixture components (likelihood) = $p(\vec{x} \mid \vec{z} = \vec{\delta}_k, \theta)$

posteriors (responsibilities) = $\tau_{n,k} = p(\vec{z} = \vec{\delta}_k \mid \vec{x}_n, \theta) = \frac{p(\vec{x}_n \mid \vec{z} = \vec{\delta}_k, \theta) p(\vec{z} = \vec{\delta}_k \mid \theta)}{p(\vec{x}_n \mid \theta)}$

log likelihood = $\sum_{n=1}^N \log p(\vec{x}_n \mid \pi, \mu, \Sigma) = \sum_{n=1}^N \log \sum_{k=1}^K \pi_k N(\vec{x}_n \mid \vec{\mu}_k, \Sigma_k)$

- Can't easily take derivatives of log-likelihood and set to 0.
- Not nice, seems to need gradient ascent…
- Or, can we do something else?

K-Means Clustering



- An old "heuristic" clustering algorithm
- Gobble up data with a divide & conquer scheme
- Assume each point x has a discrete multinomial vector z
- Chicken and Egg problem:

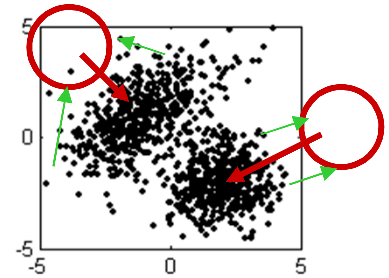
 If know classes, we can get model (max likelihood!)
 If know the model, we can predict the classes (classifier!)

- K-means Algorithm:

TIP: In practice, for EM approaches, sometimes easier to initialize z then start with an update to means.

0) Input dataset $\left\{\vec{x}_1,\ldots,\vec{x}_N\right\}$
1)Randomly initialize means $\vec{\mu}_1,\ldots,\vec{\mu}_K$
2)Find closest mean for each point $\vec{z}_n(k) = \begin{cases} 1 & if\ k = \arg\min_j \left\|\vec{x}_n - \vec{\mu}_j\right\|^2 \\ 0 & otherwise \end{cases}$
3)Update means $\vec{\mu}_k = \sum_{n=1}^N \vec{x}_n \vec{z}_n(k) \Big/ \sum_{n=1}^N \vec{z}_n(k)$
4) If any z has changed go to 2

K-Means Clustering

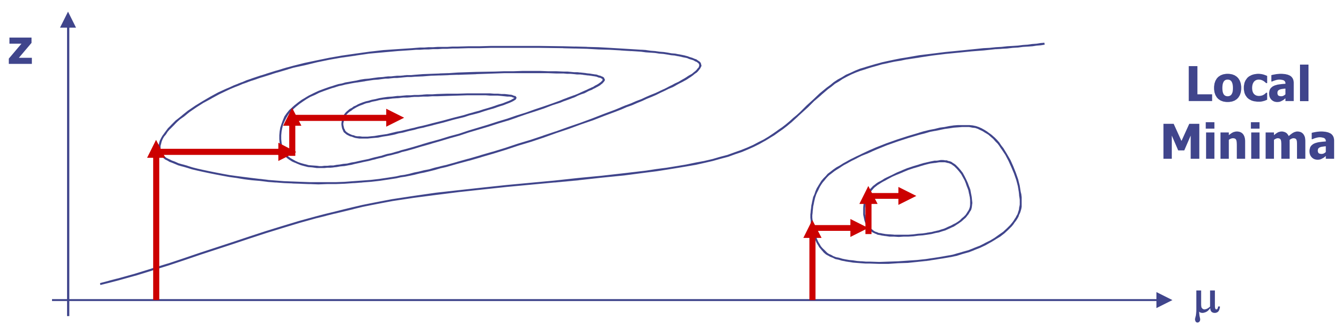


- Geometric, each point goes to closest Gaussian
- Recompute the means by their assigned points
- Essentially minimizing the following cost function:

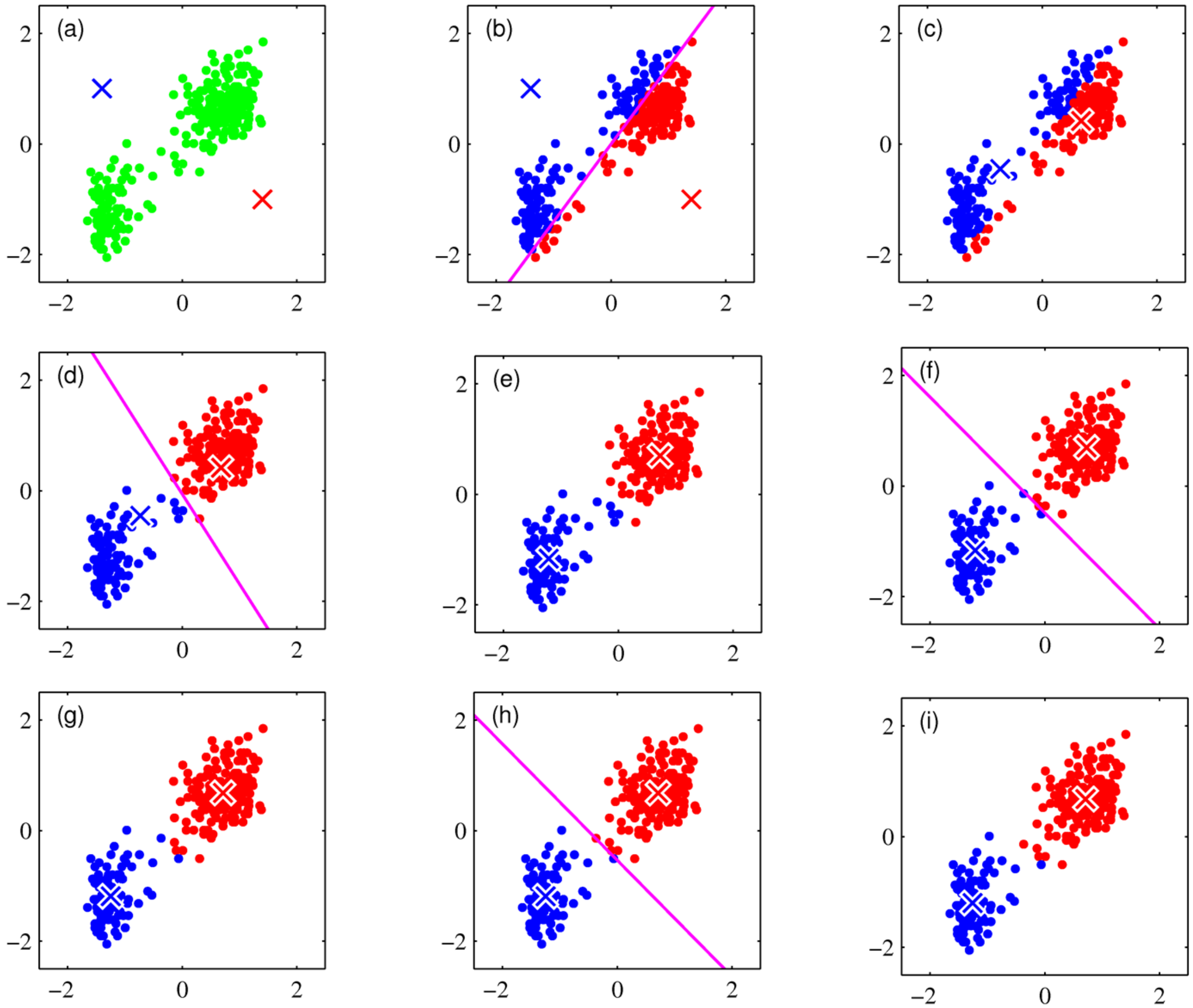
$$\min_{\mu} \min_{z} J\left(\vec{\mu}_1,\ldots,\vec{\mu}_K,\vec{z}_1,\ldots,\vec{z}_N\right) = \sum_{n=1}^{N}\sum_{k=1}^{K} \vec{z}_n\left(k\right)\left\|\vec{x}_n - \vec{\mu}_k\right\|^2$$

$$\vec{z}_n\left(k\right) = \begin{cases} 1 & if\ k = \arg\min_j \left\|\vec{x}_n - \vec{\mu}_j\right\|^2 \\ 0 & otherwise \end{cases} \qquad \vec{\mu}_k = \frac{\sum_{n=1}^{N} \vec{x}_n \vec{z}_n\left(k\right)}{\sum_{n=1}^{N} \vec{z}_n\left(k\right)}$$

- Guaranteed to improve per iteration and converge
- Like Coordinate Descent (lock one var, maximize the other)
- A.k.a. Axis-Parallel Optimization or Alternating Minimization



Example: K-means using Old Faithful data set



X marks show μ_k locations

(a) Initialization

(b) First E step

(c) First M step

… to convergence

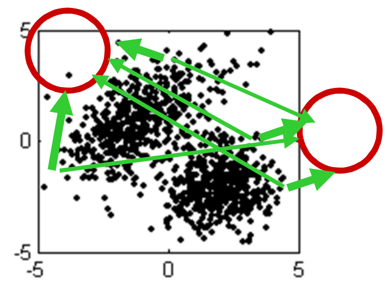
Expectation-Maximization (EM)

- EM allows a soft/fuzzy version of K-Means (winner-takes-all, closest Gaussian Mean completely wins datapoint)

$$\vec{z}_n(k) = \begin{cases} 1 & if\ k = \arg\min_j \left\|\vec{x}_n - \vec{\mu}_j\right\|^2 = \arg\max_j N\left(\vec{x}_n \mid \vec{\mu}_j, I\right) = \arg\max_j p\left(\vec{x}_n \mid \vec{\mu}_j\right) \\ 0 & otherwise \end{cases}$$

- Instead, consider soft percentage assignment of datapoint

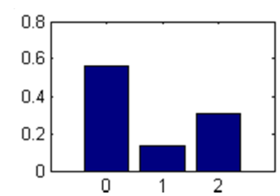
$$assign \propto \pi_k \frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}\left\|\vec{x}_n - \vec{\mu}_k\right\|^2\right)$$



- EM is 'less greedy' than K-Means uses $\tau_{n,k} = p\left(\vec{z} = \vec{\delta}_k \mid \vec{x}_n, \theta\right)$ as shared responsibility for $\vec{x}_n$

For each data point x_n, how much 'responsibility' is claimed by each class.

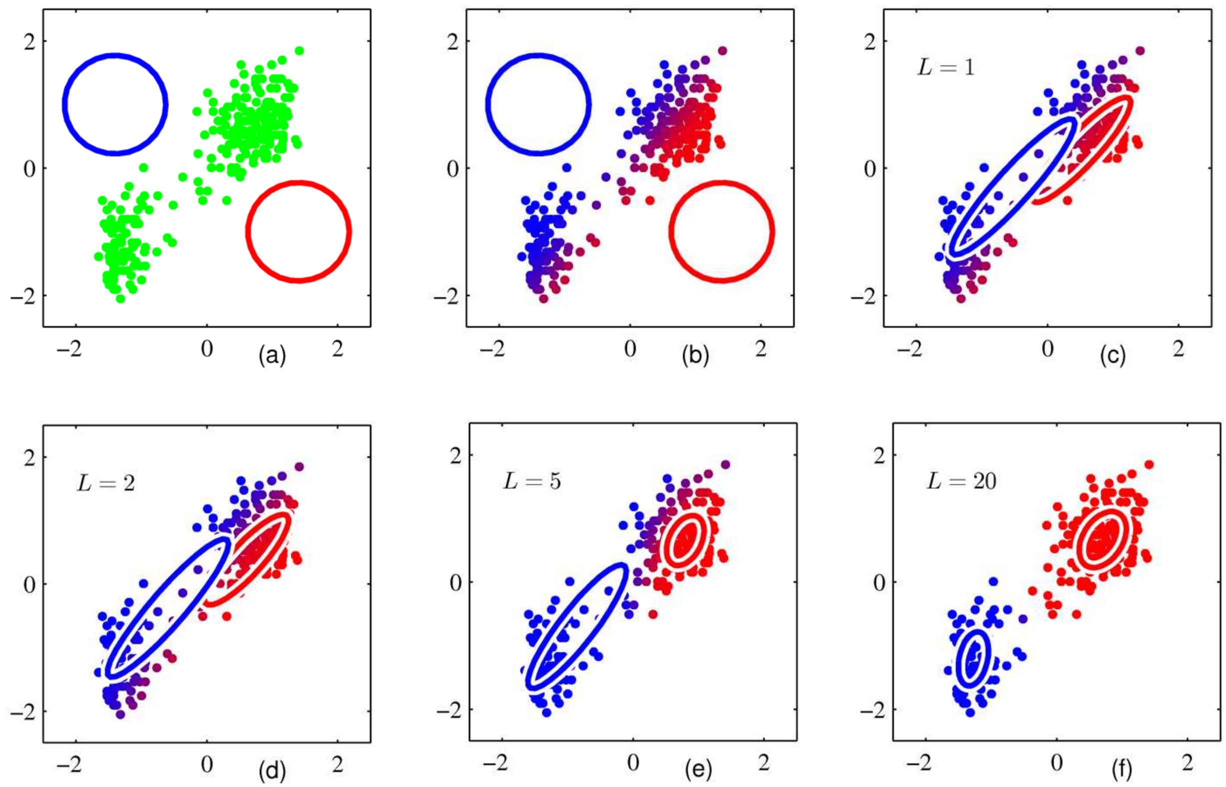
$$\tau_{n,1}, \ldots, \tau_{n,K} =$$



- Update for the means are then 'weighted' by responsibilities

$$\mu_k = \frac{\sum_{n=1}^{N} \tau_{n,k} \vec{x}_n}{\sum_{n=1}^{N} \tau_{n,k}}$$

Example: EM Mixture of Gaussians using Old Faithful data set



Initialization with same μ_k
Showing 1 stdev contours
(b) First E step
(c) First M step, after L=1 complete cycle
… subsequent cycles

Note:
Typically longer convergence time, and more overhead per cycle than K-means.

How might we initialize?

Expectation-Maximization

- EM uses expected value of $\vec{z}_n(k)$ rather than max

$$\tau_{n,k} = E\left\{\vec{z}_n(k) \mid \vec{x}_n\right\} = p\left(\vec{z}_n = \vec{\delta}_k \mid \vec{x}_n, \theta\right)$$

- EM updates covariances, mixing proportions AND means…
- The algorithm for Gaussian mixtures:

EXPECTATION:
$$\tau_{n,k}^{(t)} = \frac{\pi_k^{(t)} N\left(\vec{x}_n \mid \vec{\mu}_k^{(t)}, \Sigma_k^{(t)}\right)}{\sum_j \pi_j^{(t)} N\left(\vec{x}_n \mid \vec{\mu}_j^{(t)}, \Sigma_j^{(t)}\right)}$$

MAXIMIZATION:
$$\vec{\mu}_k^{(t+1)} = \frac{\sum_n \tau_{n,k}^{(t)} \vec{x}_n}{\sum_n \tau_{n,k}^{(t)}} \qquad \pi_k^{(t+1)} = \frac{\sum_n \tau_{n,k}^{(t)}}{N}$$

$$\Sigma_k^{(t+1)} = \frac{\sum_n \tau_{n,k}^{(t)} \left(\vec{x}_n - \vec{\mu}_k^{(t+1)}\right)\left(\vec{x}_n - \vec{\mu}_k^{(t+1)}\right)^T}{\sum_n \tau_{n,k}^{(t)}}$$

- Neat demo… http://www.cs.cmu.edu/~alad/em/
- Makes intuitive sense, but what can we prove?

Expectation-Maximization

- EM uses expected value of $\vec{z}_n(k)$ rather than max

$$\tau_{n,k} = E\left\{\vec{z}_n(k) \mid \vec{x}_n\right\} = p\left(\vec{z}_n = \vec{\delta}_k \mid \vec{x}_n, \theta\right)$$

- EM updates covariances, mixing proportions AND means…
- The algorithm for Gaussian mixtures:

EXPECTATION: $$\tau_{n,k}^{(t)} = \frac{\pi_k^{(t)} N\left(\vec{x}_n \mid \vec{\mu}_k^{(t)}, \Sigma_k^{(t)}\right)}{\sum_j \pi_j^{(t)} N\left(\vec{x}_n \mid \vec{\mu}_j^{(t)}, \Sigma_j^{(t)}\right)}$$

We'd like the true probability $p(z \mid x, \theta)$

Instead we use an approximation $q_t(z) = p(z \mid x, \theta_t)$

Need a way to think about the difference between p and q_t

- Neat demo… http://www.cs.cmu.edu/~alad/em/
- Makes intuitive sense, but what can we prove?

Entropy & KL Divergence

- Step back, reconsider Entropy, introduced in 9:16
- We'll extend the idea to Relative Entropy of 2 variables
- We'll also need Jensen's Inequality, see 11:7

- Let h(x) be the information content of an event x
- We'd like: if x and y unrelated, then h(x,y)=h(x)+h(y)
- Two unrelated events ~ independent, so p(x,y)=p(x)p(y)
- Leads to

$$h(x) = \log \frac{1}{p(x)} = -\log p(x) \geq 0$$

units?
base 2 *bits*
base e *nats*

- For a discrete random variable X, its entropy is the average information content

$$H(X) = \mathbb{E}_{p(x)}[h(x)] = \sum_x p(x) \log \frac{1}{p(x)} = -\sum_x p(x) \log p(x)$$

Entropy Properties

- Loosely speaking, H(X) is an ε-achievable lower bound on the average code rate (Shannon noiseless coding theorem)

- Example:
  - Variable X has 8 states, all equally likely
  - What's H(X) in bits?

- Example: (Cover & Thomas, 1991)
  - Variable Y has 8 states, probabilities $\frac{1}{2},\frac{1}{4},\frac{1}{8},\frac{1}{16},\frac{1}{64},\frac{1}{64},\frac{1}{64},\frac{1}{64}$

$$H(X)=\frac{1}{2}\log_2 2+\frac{1}{4}\log_2 4+\ldots$$

$$=\frac{1}{2}.1+\frac{1}{4}.2+\frac{1}{8}.3+\frac{1}{16}.4+\frac{4}{64}.6=2$$

Possible code	0	10	110	1110	{111100,111101,111110,111111}

Example: letters in English

The entropy of a randomly selected letter in an English document is about 4.11 bits (Mackay 2.4)

Compare $\log_2 27 \approx 4.75$

i	a_i	p_i	$h(p_i)$
1	a	.0575	4.1
2	b	.0128	6.3
3	c	.0263	5.2
4	d	.0285	5.1
5	e	.0913	3.5
6	f	.0173	5.9
7	g	.0133	6.2
8	h	.0313	5.0
9	i	.0599	4.1
10	j	.0006	10.7
11	k	.0084	6.9
12	l	.0335	4.9
13	m	.0235	5.4
14	n	.0596	4.1
15	o	.0689	3.9
16	p	.0192	5.7
17	q	.0008	10.3
18	r	.0508	4.3
19	s	.0567	4.1
20	t	.0706	3.8
21	u	.0334	4.9
22	v	.0069	7.2
23	w	.0119	6.4
24	x	.0073	7.1
25	y	.0164	5.9
26	z	.0007	10.4
27	-	.1928	2.4
$\sum_i p_i \log_2 \frac{1}{p_i}$			4.1

Entropy Properties

For discrete X

- H(X) ≥ 0
- H(X) = 0 iff exists some value y s.t. X=y a.s
- If X takes finite n possible values, then H(X) ≤ log n with equality iff X is uniformly distributed (maximum entropy)

For continuous X, define differential entropy

$$H(X) = -\int p(x) \log p(x) dx$$

- Note now H(X) need not be positive (e.g. consider U[0,a])
- For given mean and variance, distribution with maximum entropy is a Gaussian

KL Divergence

- Suppose have a probability distribution p(x)
- We'll approximate it with some distribution q(x)
- Consider coding scheme using q(x): information content based on q(x) but average over the true distribution p(x)
- Hence minimum average additional information required to specify x is

$$-\int p(x)\log q(x)dx - (-\int p(x)\log p(x)dx) = \int p(x)\log\frac{p(x)}{q(x)}dx$$

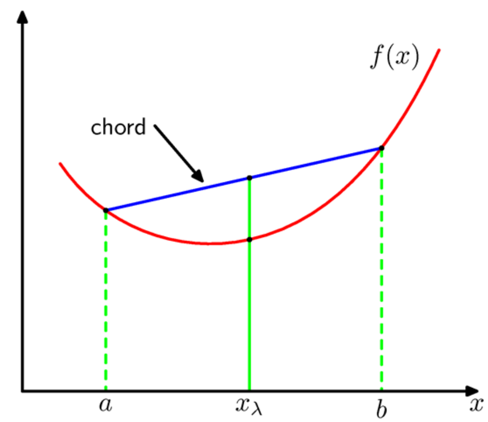
$$=: KL(p \,||\, q)$$

- Kullback-Leibler divergence or relative entropy between distributions p(x) and q(x), continuous or discrete
- Not symmetric but provides a notion of distance

Key result: KL Divergence ≥ 0

- Recall Jensen's Inequality
  - For convex f, $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$

convex f



  - Apply to KL divergence:

$$KL(p \| q) := \int p(x) . -\log \frac{q(x)}{p(x)} dx$$

-log is strictly convex

$$\geq -\log \int p(x) \frac{q(x)}{p(x)} dx = 0$$

  - Holds for discrete or continuous variables
  - Equality iff q(x) = p(x) almost everywhere