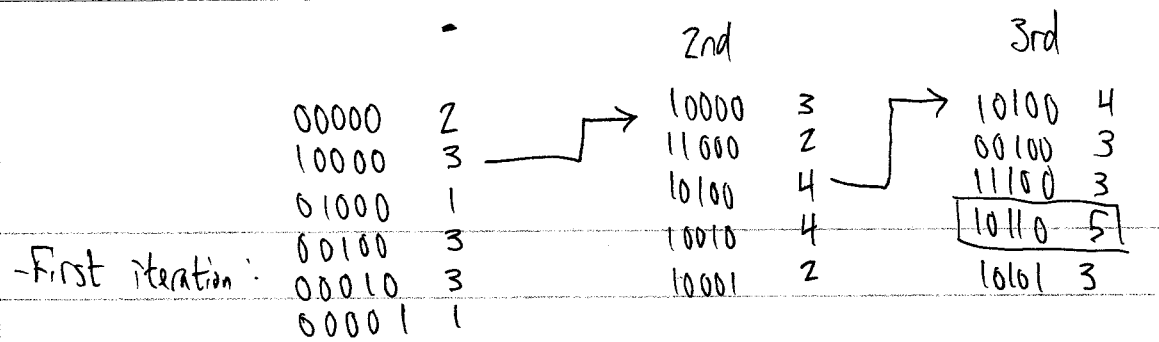
2016-02-15c, CS7641

ULI lectures

- Randomized Optimization
  - Optimization:
    - Input space X
    - Objective (fitness) function $f: X \to \mathbb{R}$
    - Goal: Find $x^* \in X$ s.t. $f(x^*) = \max_x f(x)$ (basically argmax or an approx.)
    - This comes up in many places. (Neural network training is one. Pretty much any learning algorithm with parameters fits here.)
  - Sometimes, you may simply exhaust an input space if small enough. Calculus might be able to solve analytically (if function has derivative), Newton's method works in some cases too.
  - But with a big input space, complex function, and no clear derivative, these don't work
- One algorithm: Hill climbing.
  - Guess $x \in X$.
  - Repeat:
    - Let $n^* = \underset{n \in N(x)}{\text{argmax}}\ f(x)$, N = some neighborhood
    - If $f(n^*) > f(x)$, $x = n$
    - Else: stop
  - This looks for steepest ascent but tends to get stuck in local maxima.
- e.g. X: 5-bit sequences, $f(x)$ = # of correct bits, $N(x)$ = one bit differences from x.



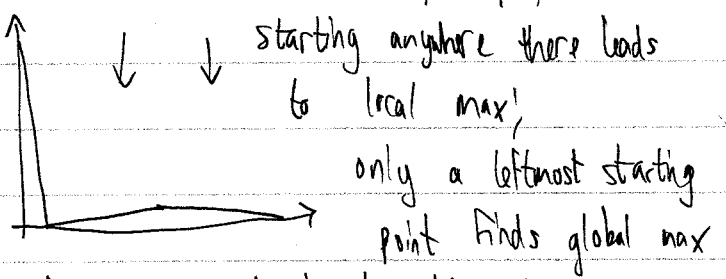
-First iteration:

- Searching from 10110 would have all neighbors worse (10110 is here the solution)
- But, this is a simpler function (no local optima)

-Random Restart Hillclimbing tries to solve this

- Once local optima reached, try again at randomly chosen x.
- This has the advantage of being more expensive only by a constant factor.
- This may require care to 'cover' the whole input space

e.g. this function:



- The size of attraction basin around local optima has a lot of influence over how long this takes on average.

-Simulated Annealing (named after blacksmithing/forge process)

exploiting vs. exploring

- Takes this 'random restart' a little further.
- Rather than always trying to improve, sometimes it tries to explore/search in a 'downward' direction.
- Isbell treats 'exploiting' too much as akin to overfitting (believing data too much),
- Related to Metropolis-Hastings algorithm

2016-02-15a, CS7641

UL1 (cont.)

- Annealing algorithm
  1. Finite set of iterations: → neighborhood
     a. Sample new point x_t in $N(x)$
     b. Jump to new sample with probability given by an acceptance probability function $P(x, x_t, T)$.
     c. Decrease temperature T. $(T > 0)$

$$P(x, x_t, T) = \begin{cases} 1 & \text{if } f(x_t) > f(x) \\ e^{\frac{f(x_t)-f(x)}{T}} & \text{otherwise} \end{cases}$$

(P tells us whether to jump to x_t or not.)

- Always hill-climb when we can
- If not, maybe move, maybe not...
  If $f(x_t) \ll f(x)$ then $P(\ldots) \approx 0$.
- Larger T: Much higher probability of 'jump'
  Smaller T: Less likely to jump (magnifies $f(x_t)$ being lower than $f(x)$)
- $T \to 0$: Like hill climbing
  $T \to \infty$: Random walk, doesn't even see 'smaller' valleys
- In practice, decrease T slowly.

Boltzmann Distribution (ish)

- $\Pr(\text{ending at } x) = \frac{e^{f(x)/T}}{Z_t}$ → That is, higher fitness has higher probability!
  → Z_t is just normalization
  - Note the $\frac{1}{T}$ factor. $T \to \infty$ means that probability becomes even.

Note similarities with random restart hill climbing (except crossover)

- Genetic Algorithms
  - Population of individuals
  - Mutation → local search in neighborhood
  - Crossover → population holds information
    - This has no real analogue elsewhere
  - Generations → iterations of improvement (hopefully)

- GA Skeleton
- P_0 = initial population of k size
- Repeat until converged {
  - Compute fitness of all $x \in P_t$
  - Select 'most fit' individuals
    (truncation selection, roulette wheel)
  - Pair up individuals, replacing 'least fit' individuals via crossover/mutation
  }
  - That 'select' step can vary. Exploiting vs. exploring comes in here somewhat — do we choose individuals randomly, weighted by fitness? Do we rank by fitness and truncate? (One can also put Boltzmann-like distributions here, as in simulated annealing, and with a temperature we vary.)
- Crossover example with X = 8-bit strings

01101100	→	01100111	Swap out groups of bits
11010111		11011100	from start → one-point crossover

  - This suggests an inductive bias:
    - Locality matters in bits (we flip only in adjacent groups)
    - There are independent 'subspaces' to optimize (groups of bits)
  - Perhaps we want locality to matter less, and adjacent bits to split up more.
  - Then just flip-flop bits at random:

01101100	→	01000110	→ Uniform crossover
11011011		11111101	(this happens at the level of real genes actually.)

  - Genetic algorithms are "the second-best solution to any given problem" (sounds like a back-handed compliment to me), and useful in many contexts

2016-02-15e, CS7641

ULI (cont.)

- The 'randomized' in randomized optimization refers to the random steps we take and to starting in random places (in order to avoid stepping a gradient that points the 'wrong' way)
- They are optimization but they come up frequently in learning (anytime parameters of a model need tuning).
- Those methods (hillclimbing, simulated annealing, GAs), though, all don't "remember" much. They just keep track of where they are, not where they've been. They track a single "point" (or population), but as that evolves, they throw away everything about that function's structure/behavior.

"amnesic"

- So: What out there keeps track of points, history, or the 'optimization space', or tries to capture the probability distribution we model (maybe implicitly)?
- One is TABU search - keeps track of areas already seen, & avoids them in the future in order to explore the space better.

2016-02-16

Isbell wrote a paper on this ~20 years ago

- MIMIC
  - Idea is to directly model probability distribution as one does a search through space, and that as this model is refined it begins to reveal structure

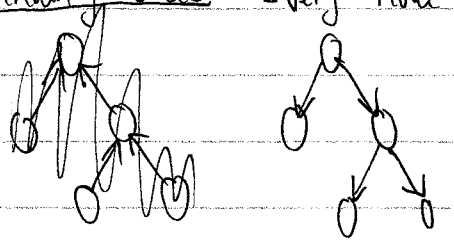
- Mimic's prob. distribution:

$$P^{\theta}(x) = \begin{cases} 1/z_\theta & \text{if } F(x) \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

θ = threshold
F = fitness
z_θ is just normalization
So P is uniform for all $f(x)$ over threshold! Below threshold, it's 0.

$P^{\theta_{min}}(x)$ = uniform
$P^{\theta_{max}}(x)$ = optima only

- Start out with $P^{\theta_{min}}(x)$ - the uniform distribution across the whole space. The goal is to end at $P^{\theta_{max}}(x)$ - at which point we simply sample from optima
- Pseudo code:
  - Generate samples from $P^{\theta_t}(x)$
  - Set θ_{t+1} to n_{th} percentile (like GAs, sort of)
    Retain only samples s.t. $f(x) \geq \theta_{t+1}$ (also, sort of similar to particle filters)
  - Estimate $P^{\theta_{t+1}}(x)$
  - Repeat (until we reach θ_{max}, or whatever)
- The structure then is in $P^{\theta_t}(x)$
- But how do we estimate $P^{\theta_{t+1}}(x)$?
  Note though that samples in $P^{\theta_{t+1}}(x)$ are also in $P^{\theta_t}(x)$?
- Suppose x has n features, $x = [x_1, x_2, x_3 \ldots x_n]$
  $p(x) = p(x_1 | x_2 \ldots x_n)\, p(x_2 | x_3 \ldots x_n) \ldots p(x_n)$
  → The probability of seeing x is just joint distribution over all features.
  - Normally we'd need a huge amount of data to estimate early terms. We can make some independence assumptions though.
  - With dependency trees: Every node has one parent.



(Every random variable depends on exactly one random variable)
(Except overall parent)

Contrast w naive Bayes in which every node has the same parent.

2016-02-16(b), CS7641

ML1 (cont.)

- Then, $\hat{P}_\pi(x) = \prod p(x_i \mid \pi(x_i))$ ∴ where π is parent. And $\pi(\text{parent}) = \text{parent}$
  - Joint distribution can be rewritten thus
- Algorithm needs only quadratic ~~space~~ data?, but also must determine correct tree. The only real question there is just 'how many relationships?'
- Dependency trees let you represent relationships, - but limit how many of them. It avoids interrelationships/covariance between variables. Really, it's the simplest that doesn't assume independence. (e.g. $\prod p(x_i)$ assumes all variables are independent.)
- It lets one represent similar relationships as crossover in GAs. Crossover represents structure with locality, and this is a general form of that (it no longer needs locality).
- Also, these trees are very easy to sample from: It's just topological order, starting at root, linear in # of features.
- So: How do we find dependency trees?
  - We have some 'true' distribution, p, which we estimate with $\hat{P}_\pi$.

→ $D_{KL}(p \,||\, \hat{P}_\pi) = \sum p[\lg p - \lg \hat{P}_\pi]$ - This tells how close p & $\hat{P}_\pi$ are.

- Want to minimize this with dependency tree, that is, π.
- But $p \lg p$ = entropy = $h(\ldots)$

$D_{KL}(\ldots) = -h(p) + \sum h(x_i \mid \pi(x_i))$ - So, entropy + conditional entropies

- But $h(p)$ is independent of π, ignore it.

Let $J_\pi = \sum h(x_i \mid \pi(x_i))$ - We want to minimize this.

- Basically, best tree minimizes all entropy for each feature, given parents.

D_{KL} = Kullback-Leibler Divergence (information theory) Not a distance, a divergence.

$h(\ldots|\ldots)$ = conditional entropy

- That is, we want parents to give the best information, such that knowing parent tells you the most about whatever descendant.
- But, J_π is a pain to minimize.

So, let $J'_\pi = \underbrace{-\sum h(x_i)}_{\text{That is independent of } \pi \text{ and thus doesn't matter for minimization.}} + \sum h(x_i | \pi(x_i))$

But $J'_\pi = -\sum_i I(x_i, \pi(x_i))$ where I is <u>mutual information</u>

Or, maximize $\sum_i I(x_i, \pi(x_i))$ from information theory.

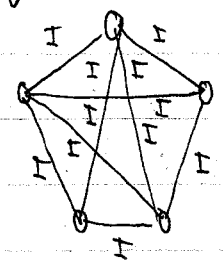
→ Maximize <u>mutual information</u> between x_i & $\pi(x_i)$! (that is, feature & parent)

Conditional entropy is directional. Mutual information is not!

This turns out to be much easier.

- That equation actually induces a fully connected graph over each node:

$O(n^2)$ links



where each 'I' represents mutual information between those nodes.

We want to find the tree that maximizes mutual information and is embedded in that graph's connections.

- That is, we want the <u>maximum spanning tree.</u>

(or, min. spanning tree & negate edges). Use Prim because this is a densely connected graph (?) and it runs in $O(n^2)$.

- Then pick any node as your root, and there's your optimal dependency tree.
- Then... finally $P^{\theta_{opt}}_{(x)}$ is estimated (look a few pages ago). Merely building samples has meant we already have all the conditional probabilities (which we need for entropy)
- MIMIC doesn't <u>have</u> to use dependency trees. (you could use something much more complex), but they work very well.

2016-02-16c, CS7641

UL1 (cont.)

- MIMIC doesn't care about the 'actual' underlying distribution of features.
- But, dependency trees easily 'overfit' independent distributions (it takes a lot of data to show that conditional probability tables are sort of meaningless).
- Practical matters w/ MIMIC
  - Does well with structure. Multiple optima don't confuse it.
  - Represents $P^{\theta}\ \forall\theta$
    - It can be stuck in local optima but isn't as susceptible
  - Involves probability theory!
  - The price: Time?
    - Isbell says that it takes orders of magnitude fewer iterations than things like simulated annealing to get 'right' answer.
    - But, those are likely slower iterations because of what else is involved. At the same time, that's more information at every iteration.
    - It works well when cost of $F(x)$ is high, though. It reuses samples, θ_t generates samples for θ_{t+1}, for instance.
    - That's whether it's a high <u>computational</u> cost - or whether $F(x)$ involves humans running some test.

2016-02-24

- Notes on: http://scott.fortmann-roe.com/docs/BiasVariance.html
- Prediction errors can be due to bias or due to variance.
- Much of this only makes sense in the context of multiple models over different data.
- Bias: Difference between expected/average prediction of model, and correct value we are trying to predict.
- Variance: Variability of a model prediction for a given data point.
- Suppose $Y = f(X) + \epsilon$, $\epsilon \sim N(0, \sigma_\epsilon)$, $\hat{f}(x)$ estimates $f(x)$.

$$Err(x) = E\left[\left(Y - \hat{f}(x)\right)^2\right] = \underbrace{\left(E\left[\hat{f}(x)\right] - f(x)\right)^2}_{\text{Bias}^2} + \underbrace{E\left[\left(\hat{f}(x) - E\left[\hat{f}(x)\right]\right)^2\right]}_{\text{Variance}} + \underbrace{\sigma_\epsilon^2}_{\text{irreducible error}}$$

  - With a true model & infinite data we can bring both bias & variance to 0, but not irreducible data.
  - With imperfect models & finite data, we must trade off between minimizing bias, and minimizing variance.
- With kNN, for instance, lower k → higher variance, lower bias
  higher k → lower variance, higher bias

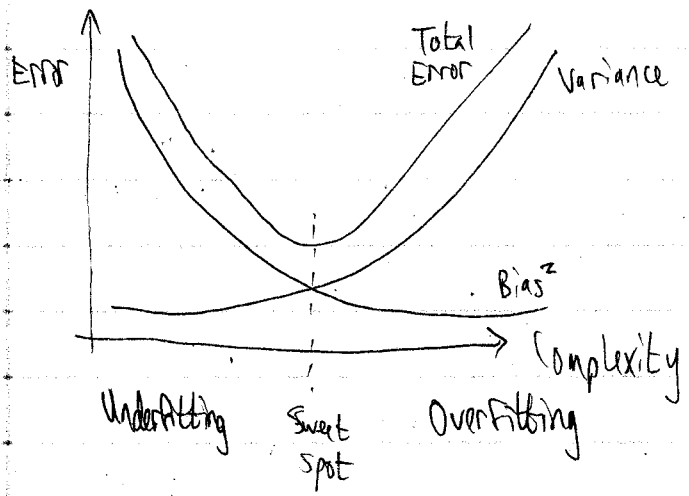
$$Err(x) = \underbrace{\left(f(x) - \frac{1}{k}\sum_{i=1}^{k} f(x_i)\right)^2}_{\text{Bias}^2} + \underbrace{\frac{\sigma_\epsilon^2}{k}}_{\text{Variance}} + \underbrace{\sigma_\epsilon^2}_{\text{irreducible error}}$$

- Bagging (bootstrap aggregation) is a good way to reduce variance.
- asymptotic consistency: training size → ∞, bias → 0
- asymptotic efficiency: " " ", variance → no worse than any other model

2016-02-24(b), CS7641

Bias-Variance Tradeoff (continued)

- These are good properties to have, but they don't say anything about behavior with smaller training sizes.
- More complex model: Lower bias, higher variance



At that sweet spot:

$$\frac{d\,\text{Bias}}{d\,\text{Complexity}} = -\frac{d\,\text{Variance}}{d\,\text{Complexity}}$$

- This link also recommends 'Elements of Statistical Learning' (Springer text - Hastie, Tibshirani, Friedman)

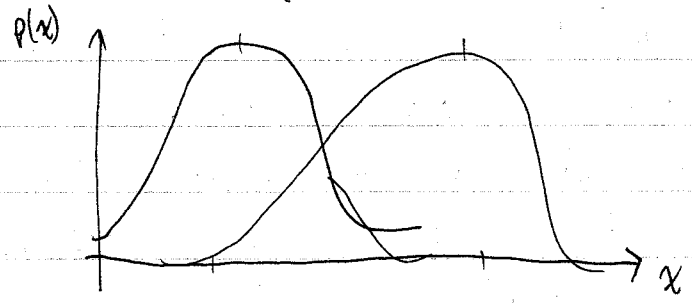
2016-03-02 for a logical order

~~2016-02-12~~, CS7641

Mitchell
Ch. 6 (cont)

- Monte Carlo methods can approximate by sampling
- But how do we <u>learn</u> Bayesian networks, whether with network structure given in advance or not?
  - Former is straightforward if all variables are fully observable.
  - It is more difficult if only some variables are observable.
- Gradient ascent is one means. (6.11.5)
  - Tries to maximize $P(D|h)$ for hypothesis space consisting of all possible entries for the conditional probability tables (note that this assumes a given structure).
  - It is similar to learning weights of hidden units in ANN (because training data will give inputs & outputs - but not the values of hidden units)
- Learning the <u>structure</u> of networks is more difficult (I'm going to assume that's unchanged since 1997).
- EM algorithm
  - Widely used to learn in the presence of unobserved, unobservable, partially observed variables. Only the general form of their probability distr. needs to be known.
  - Can train Bayesian networks, but also radial basis functions, and is the basis for much unsupervised clustering and the Baum-Welch forward-backward algorithm for learning Partially Observable Markov Models

- Suppose we have x_i, a random variable whose value is chosen at random from <u>two</u> normal distributions with same (known) σ^2 but different means. (unknown)



e.g. from the two on this plot

- How would we determine a hypothesis $h = \langle \mu_1, \mu_2 \rangle$ that is maximum-likelihood for these means? (Note that if we had only a single normal distribution, it is easy to find this - it's just the sample mean, $\mu_{ML} = \frac{1}{m} \sum_{i=1}^{m} x_i$)
- Here we have a 'hidden variable': Which distribution produced which instances. We might think of an instance not as x_i but $\langle x_i, z_{i1}, z_{i2} \rangle$ where $z_{i1} = 1$ if 1st distribution, 0 otherwise, and likewise for z_{i2} & 2nd. If we knew z_{i1} \ z_{i2} we could find means directly.
- This generalizes to k-means rather than just 2 means. EM works by repeatedly:
  - Re-estimating expected values of z_{ij} given $\langle \mu_1 \cdots \mu_k \rangle$
  - Re-calculating max. likelihood hypothesis using z_{ij}

$\mu_1 \cdots \mu_k$ are initialized to arbitrary values typically

- For two means it is: $E[z_{ij}] = \frac{p(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i | \mu = \mu_n)} = \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$

  - to estimate, substituting $\langle \mu_1, \mu_2 \rangle$ and observed x_i

$\mu_j \leftarrow \frac{1}{m} \sum_{i=1}^{m} E[z_{ij}] x_i$ - to re-calculate.

  - This is just weighted sample mean.

2016-03-02b

~~2016-02-18e~~, CS7641

Mitchell
Ch. 6 (cont)

- EM applies more generally to estimating any parameters θ that describe an underlying distribution, given only observed data.
- Let $X = \{x_1 \dots x_m\}$, observed data in 'm' independently drawn instances. Let $Z = \{z_1 \dots z_m\}$, unobserved data of the same. Let $Y = X \cup Z$, full data.
  - X, Y, and Z all can be treated as random variables depending on prob. distr. with parameters θ (and Z also depends on X)
  - h is 'current' hypothesized value of θ; h' is 'revised' hypothesis each iteration refines.
- EM tries to find h' to maximize $E[\ln P(Y|h')]$
  - Note that it's over full data Y, but we also don't know the 'real' probability distribution of Y (we're trying to find it!)
    - h' supplies the parameters θ that we assume for Y
- More Formally:

Estimation

1. E: Calculate $Q(h'|h)$ w/ current h and X to estimate prob. distr. over Y: $Q(h'|h) \leftarrow E[\ln P(Y|h')|h, X]$

Maximization

2. M: Replace h with h' that maximizes Q:
   $h \leftarrow \underset{h'}{\operatorname{argmax}}\, Q(h'|h)$

- Always converges to a single maximum (if only one) but may converge to local maxima too.
- k-means is actually a special case of EM for k normal distributions
  (is this the same k-means as in clustering?)

General product rule/chain rule:

$$P(X_1, X_2, \ldots X_n) = P(X_1 | X_2, \ldots X_n)\, P(X_2 | X_3 \ldots X_n) \ldots P(X_{n-1} | X_n)\, P(X_n)$$

$$Pr(D) = \sum_{h \in H} P(D|h)\, P(h)$$

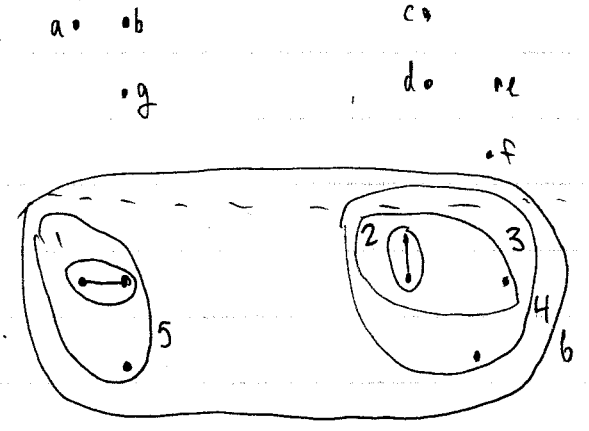
$$= \frac{|VS|}{|H|}$$

$$Pr(D|h) = \prod_{i} Pr(d_i | h)$$

2016-03-02c, CS7641

UL-2

- Clustering & Expectation Maximization
- Thus far, we did supervised learning - using labeled training data to generalize labels to new instances; function approximation.
- Or "compact representation"
- Unsupervised learning is more data description, making sense of unlabeled data.
- Basic clustering problem (a classic unsupervised problem):
  - Given: X, a set of objects → Need not be metric space or even a true distance function
    D, inter-object distance, $D(x,y) = D(y,x)$ for $x, y \in X$
  - Output: Partition $P_D(x) = P_D(y)$ if x & y in same cluster
  - Trivial answer: $\forall x, P_D(x) = 1$ (one big partition)
    Or: $\forall x, P_D(x) = x$ (one partition per object)
    - But, not very useful.
- D can just be a matrix, even.
- More useful: single linkage clustering
  - Consider each object a cluster (n objects)
  - Define intercluster distance as distance between closest two points in the two clusters
  - Merge two closest clusters
  - Repeat n-k times to make k clusters
  - e.g. If we have:



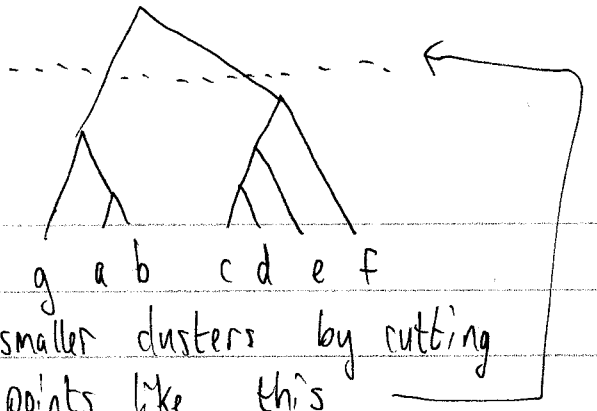
- This is hierarchical agglomerative clustering (HAC)
- It may iterate like...

We could think of many meaningful ways to cluster, depending on cluster size. This captures that.

Or, end before 6.

median = "non-metric" statistic (numbers don't matter)

mean is a metric statistic



- We can represent like: g a b c d e f
  - and we may get bigger/smaller clusters by cutting tree at different points, like this
- If we use something besides closest distance, we can get something meaningful (e.g. average/median) but it is not single-link clustering
- Nice property of it: → Perhaps lower with better distance storage
  - It is deterministic (and also $O(n^3)$ for n points)
  - If your distance represents edge lengths of a graph then this is just minimum spanning tree
- Problem with single-link clustering: It will tend to cluster big 'strings' together first, where individual distances are low.
- A better way is k-means clustering:
  - Pick k 'centers' at random. (From points)
  - Each center 'claims' its closest points.
  - Recompute centers by averaging clustered points
  - Repeat until convergence. (e.g. clusters do not change)

I think we did this in CS6476

This center need not be one of the points

Video #7 has a visual demo of this

- k-means in Euclidean space:

  $p^t(x)$: Partition/cluster of object x

  c_i^t : Set of all points in cluster $i = \{x \mid P(x) = i\}$

  $\text{center}_i^t = \sum_{y \in c_i^t} y / |c_i|$

  - Note that this is mean or centroid.

  k of them - hence k means

Or, iteratively:

$$\text{center}_i^0 \rightarrow p^t(x) = \arg\min_i \|x - \text{center}_i^{t+1}\|_2^2 \rightarrow \text{center}_i^t = \sum_{y \in c_i^t} y / |c_i|$$

Euclidean distance

$t = t+1$

This is a sort of successive optimization as it loops here

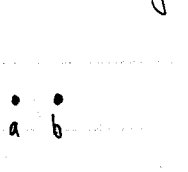
2016-03-02d, CS7641

UL-2 (cont.)

- k-means as optimization
  - With optimization we had — configurations - center, P
    - scores - some error of P(center
    - neighborhoods - P, center = $\{(P', center)\} \cup \{(P, center')\}$ (Change one at a time)
  - That error:
    $$E(P, center) = \sum_x \left\| center_{P(x)} - x \right\|_2^2$$
  - In this sense, k-means is just moving around in that space trying to minimize that error - which looks a lot like hill-climbing, except that we haven't shown yet that k-means picks the neighboring value that maximizes score
- Look on prior page for $P^t(x)$ definition. We are looking for the center which minimizes squared distance to each x. We 'move' a point only if that distance decreases: So either we decrease E or it doesn't change.
- Now see $center_i^t$: It is the average, which always gives the minimum squared error assuming partitions don't change.
- N.B. We have finite labels and finite points, thus finite configurations (even if our space is continuous, centers are combinations of the points), thus some limit to how long it can iterate.
  - If we have a way to break ties so that we don't simply go in a loop that returns to prior configs., then we're guaranteed convergence, in finite time.
    There are k^n configurations, which is sort of useless, but in practice we need nowhere near this.

break ties consistently, e.g. lexicographically

- k-Means properties:
  - Each iteration is $O(kn)$ or $O(dkn)$ with d dimensions.
  - $O(k^n)$ max iterations
  - Error decreases if we break ties consistently (except if we've converged fully).
  - But: Can get stuck! Consider:



    - Optimal clusters are a|b, c|d, e|f.
    - But what if starting centers are a, b, and c?
    - Then it is stuck with: a, b, c|d|e|f.
  - These are 'local optima' (which hill climbing is susceptible to). We can random-restart, or try to pick "spread-out" starting clusters

Or "soft k-means"

- Soft clustering
  - Consider: a b c d e f g
  - With k-means, what happens to 'd'? (Assume equidistant from c & e)
    - It would end up with a|b|c or with e|f|g depending on starting conditions. If they could sort of 'share' d...
  - For soft clustering, assume data was generated by:
    1. Select one of k Gaussians (fixed, known variance) uniformly
    2. Sample x_i from that Gaussian.
    3. Repeat n times.
  - Task: Find hypothesis $h = \langle \mu_1, \ldots \mu_k \rangle$ that maximizes probability of the data (ML) → means (k of them... not k-means though)
  - Note that the maximum likelihood mean of one Gaussian is just... the mean of the data.

2016-03-02e, CS7641

UL-2 (cont.)

- But what if we have k of them? This can be seen as hidden variables, e.g. $\langle x, z_1, z_2, \ldots z_k \rangle$ where $z_i = 1$ if x came from Gaussian i (otherwise 0)

(EM) → Expectation Maximization (actually fairly similar to k-means)

$$E[z_{ij}] = \frac{P(x=x_i \mid \mu=\mu_j)}{\sum_{i=1}^{k} P(x=x_i \mid \mu=\mu_j)} \qquad \mu_j = \frac{\sum_i E[z_{ij}]\, x_i}{\sum_i E[z_{ij}]}$$

(z: $E[z_{ij}]$ → μ_j; μ: μ_j → $E[z_{ij}]$)

Expectation (define z from μ) — Maximization (define μ from z)

No priors because this is max. likelihood. Start again at random points

- We bounce between a 'soft clustering', and recomputing means
- denominators are just normalization.

See video #17 for demo

$$P(x=x_i \mid \mu=\mu_j) = e^{-\frac{1}{2}\sigma^2 (x_i - \mu_j)^2}$$

- If we pushed P to 0 or 1 we'd end up with just k-means. k-means minimizes error, EM maximizes probability, and in the process tells us some probabilities.

- As we iterate, we produce probabilities of each point belonging to each cluster - hence 'soft clustering'. Ambiguous points are denoted as such rather than being forced to one or the other.

- EM properties
  - Monotonically non-decreasing likelihood
  - Doesn't diverge, but isn't guaranteed to converge (though does practically)
  - Can still get stuck (local optima issues as with k-means)
  - Works with any distribution (if E, M solvable), not just Gaussians

Usually, E step harder.

Look up: "method of moments estimator"
Jensen's inequality

- Clustering properties
  - Richness: For any assignment of objects to clusters, some distance matrix D exists such that P_D returns that clustering.

$\forall c \; \exists D \; P_D = c$

  - Scale-invariance: Scaling distance by positive value doesn't change clustering (or: units don't matter)

$\forall D \; \forall k > 0 \quad P_D = P_{kD}$

  - Consistency: Shrinking intra-cluster distances and expanding inter-cluster distances doesn't change clustering
- Consistency is, loosely, that tighter clusters further away are clustered the same.
- Single-link clustering, stopping at 1/2 clusters, has scale-invariance & consistency
  " " " " stopping at θ units apart, has richness & consistency
  " " " " θ/w " " has richness & scale-invariance

Can't represent smaller clusters → not rich

θ is scale-specific

If we move clusters further apart, $\theta/w \to 0$ and this changes clustering

$$w = \max_{C_i, C_j} D(i,j)$$

- It turns out that no algorithm can have all three properties.
  → Kleinberg's Impossibility Theorem
- Domain knowledge is what you actually need out of the clusters & algorithms, and what properties you need.

"An Impossibility Theorem for Clustering" (Kleinberg) – Most lecture examples follow this & its notation

- Summary (ish):
  - Clustering relates to compact description
  - Algorithms: - k-means (guaranteed to converge)
    single-link clustering (terminates fast)
    EM (allows 'soft' clusters & probability).

- Papers: "The Expectation Maximization Algorithm" (Dellaert, 2002, GIT-GVU-02-20)

2016-03-15, CS7641

UL-3

- Feature Selection (in an unsupervised setting)
  - Why do it?
    1. Knowledge discovery, interpretability & insight (Which features actually matter, of the ones we keep track of?)
       - This is more for the sake of the human, but it is very important!
    2. Curse of dimensionality - data needed grows exponentially with number of features.
       - This is for the algorithm.
  - Goal: Look at your bunch of features, feed them to an algorithm, figure out what features matter!
  - But suppose we have N features and want to find the best M < N features, given some function which tells you a 'score' of features. But, we don't know M, or anything about f.
    - This is exponential - you must check every subset of N, which is NP-hard & 3-SAT.

2^N

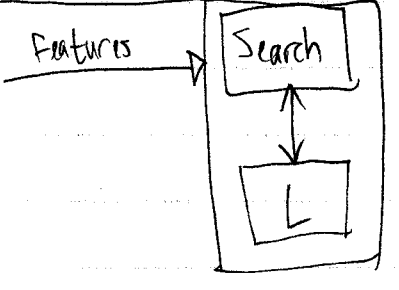
- Two broad approaches to this: filtering & wrapping.

L = some learning algorithm

- Filtering:

- Search is some algorithm (the notion of how 'good' features are is then here, with filtering

- Wrapping:



Here, the learning algorithm determines (or helps) which features

UL-3 (cont.)

- So filtering is always reducing features first. Learning algo. can't feed any information 'back' to filtering.
- In wrapping, reduction is intrinsically tied with learning algorithm and can feed info back.
- Filtering is faster, and sort of more modular. It tends to look at features in isolation though, and ignores if two features are useful together.
- Wrapping takes model bias & learning into account, and is far slower - it must run learning algorithm to determine anything.
- Note that decision trees & boosting sort of already do filtering - think of ID3. Filtering can take example labels to help with filtering.
  - Information gain does help here similarly. We could actually use a decision tree learner as the search algorithm - but don't pass the tree, just pass the relevant features used in it. We could use the inductive bias of DT to filter features, then another learner after.
- We could look at variance, entropy, Gini index, training a neural net & then pruning away low-weight inputs, removing features that are linearly dependent on others...
- How might we make wrapping more efficient?
  - Hill climbing / gradient descent (pick random subset of features, run learning, try others, see improvement... Or, really use any randomized optimization technique we've covered - SA, MIMIC, GA)

2016-03-15b CS7641

UL-3 (cont.)

- Forward search:
  - Try all features in isolation (run L on each).
  - Keep best feature. Then, try all other features in combination with this one at a time. Keep best combination, and iterate again.
  - If adding a feature gives worse result, then stop there.
- Backward search
  - Try all features at once - except one. Try removing each one separately, and see which one makes the least impact when removed. Then repeat, removing two (one from the last step - and one that alternates between all others).
- these are similar as well to hill-climbing.
- Relevance
  - x_i is strongly relevant if removing it degrades B.O.C.
    ↳ From set of all features, compared w/ features minus x_i
  - x_i is weakly relevant if:
    - not strongly relevant, and
    - $\exists$ subset of features S | adding x_i to S improves B.O.C.
  - x_i is otherwise relevant.
- Adding a feature can turn a strongly relevant feature to a weakly relevant one.
- Relevance measures effect on B.O.C. (it's about information)
- Usefulness measures effect on a particular predictor (effect on error, given a specific model/learner).

B.O.C. = Bayes Optimal Classifier

- Irrelevant feature can be useful still on some learner.
- B.O.C. is the best we can do given the information (all others have some bias); relevance is usually what information-theory metrics 'care' about.

UL-4

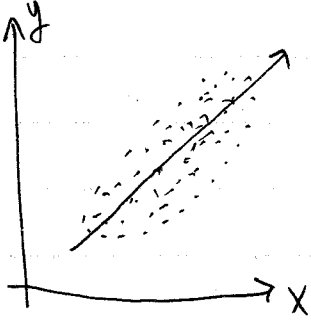
- Feature Transformation
  - The problem of pre-processing a set of features to create a new (smaller? more compact?) feature set, while retaining as much (relevant? useful?) information as possible.
    $x \sim \mathbb{F}^N \rightarrow \mathbb{F}^M$, $m < N$ (usually), P^T_x (usually) ↗ linear (P = matrix)
  - This is actually a superset of Feature selection. That is, Feature selection is a very constrained form thereof in which we only transform by choosing existing features.
  - We're also concerned here with <u>linear</u> transformations.
    - We want to 'project' our features into a smaller space, e.g. $(x_1, x_2, x_3, x_4) \rightarrow (2x_1 + x_2)$ whereas w/ selection we turned it to (x_1, x_2) for instance
    - Note that with perceptrons we already projected into a <u>higher</u> space in order to make something like XOR linearly separable, and likewise w/ SVM & kernel trick. But here we're only doing lower spaces.
  - An example to motivate 'why': Consider Information retrieval, particularly, the ad-hoc retrieval problem (a.k.a. the Google problem). You have a whole bunch of documents & want the documents relevant to a query.

2016-03-15c, CS7641

UL-4 (cont.)

- Give only some features; ad-hoc means that query isn't known ahead of time.
- But what features matter? Words?
  - One issue: Lots of words exist, and can mean multiple things, and many words mean the same thing → polysemy, → synonymy
    - Polysemy gives false positives, synonymy gives false negatives
  - Words are good indicators, but insufficient, & feature selection won't solve this. However, what could we do to combine words via some kind of feature transformation?
- First method we'll discuss: PCA, principal components analysis.
  - It's an example of an eigenproblem.

Read on this separately.



Consider the data on the left. It has two dimensions, x & y. PCA would identify the diagonal line as the first dimension.

What PCA does is find directions of largest variance, if we project onto that. (Feature selection is just projecting onto x or y here.) It also finds mutually orthogonal directions.

- That diagonal is the "first principal".
- SVD is one form of PCA (there are many).
- PCA is a global algorithm (?)

- PCA can also be proven to give the 'best' reconstruction (loses no information). It's basically a relabeling of the dimensions. It also minimizes L_2 error if we pick $N < M$ dimensions from 1st, 2nd, ... Nth principal, and reconstruct on original axes. (vs. any other reconstruction from N dimensions.)

side effect of max variance

- Because it's an eigenproblem, eigenvalues monotonically non-increase from 1st principal to 2nd, 2nd to 3rd, and onward.

If eigenvalue ⇒ then feature provided no information

- Basically: We transform into a feature space where we can easily perform feature selection.
  - the non-increasing & zero eigenvalues help here.
- Generally, we subtract mean from data too (otherwise PCA doesn't tell us as useful information)
- PCA, importantly, is well-studied & fast (with existing implementations)
- But, how's this relate to classification?
  - It's closer to a 'filter' method rather than 'wrapping'
- Independent Components Analysis
  - PCA is about finding <u>correlation</u> by maximizing <u>variance</u>, giving ability to reconstruct
  - ICA is trying to maximize <u>independence</u> (in the statistical sense) in some transformed feature space, again a linear transformation. It wants to maximize mutual information between Y and X (for transforming X to Y), while making it 0 for all pairs of features in Y.
    - We have hidden variables & observables, assumed independent, and want to figure out hidden variables

Independent: Mutual information = 0 between any pairs

2016-03-15d, CS7641

UL-4, cont.

- Standard example is 'cocktail party problem' or blind source separation

2016-03-16

- We might have...

$$\begin{bmatrix} 100 & 127 & 55 & \dots \\ 13 & 43 & 19 & \dots \\ 7 & 41 & 12 & \dots \end{bmatrix} \begin{matrix} \leftarrow \text{feature} \\ \leftarrow \text{feature} \\ \leftarrow \text{feature} \end{matrix}$$

Sample 0 1 2 ...

  - That is, each column is one sample (as in sampled audio) and each row is one recorded feature that is a linear combination of the sources.
  - We are looking for a linear combination of those (a "projection") giving us our sources back.
  - But what does 'mutual information' mean here?
    - Here it means that knowing one source doesn't help you know anything about the others
  - We can generate independent projections at will, but we don't want to lose anything in the process.

- PCA
  - Mutually orthogonal (global constraint in definition)
  - Maximal variance (it tries to make uncorrelated dimensions); "maximal reconstruction"
  - Produces ordered features

But sometimes, PCA gives same results

(when all data is Gaussian, for one thing)

- ICA
  - Mutually independent (no requirement in definition of orthogonality)
  - Maximal mutual information (not variance - see next page)
  - Produces unordered features

- PCA is trying to find uncorrelated Gaussians, basically.
- It is fundamentally a different model than ICA.
- Uncorrelated can be independent. Sometimes, max variance = independent.

- Note that a bunch of distributions, summed together, (as in a linear combination) tends to produce a Gaussian (Central Limit Theorem)
  - Maximizing variance (as in PCA) is trying to combine a bunch of independent things - not separate them!
  - ICA assumes variables are independent and non-Gaussian. Its model is such that maximizing variance is the wrong thing.
- PCA produces ordered features (as we mentioned with 1st, 2nd, etc. principals, and non-increasing eigenvalues). ICA produces unordered features, though you can sort of use kurtosis to get an order in certain cases.
- PCA and ICA both try to capture data in some other projection, but they make fundamentally different assumptions.
- ICA solves blind source separation well. PCA does very poorly.
- ICA is "directional", where PCA is not; if you transpose the input matrix, PCA finds same result.
- PCA applied to images of faces:
  - 1st principal = brightness (usually we just subtract out average)
  - 2nd principal = an average face (strange, but works for reconstruction)

"eigenfaces"

- ICA applied to same:
  - Finds nose features, eye features, hair features...
  - Since it has no global constraint on orthogonality, it finds local features. (Versus global features in PCA.)
- ICA on a nature scene finds edges. In some ways this tells us that they are fundamentally the building blocks of nature scenes.
- ICA finds topics with information retrival & documents.
- In both cases you can compute these much more readily without ICA.

2016-03-16, CS7641

UL-4 cont.

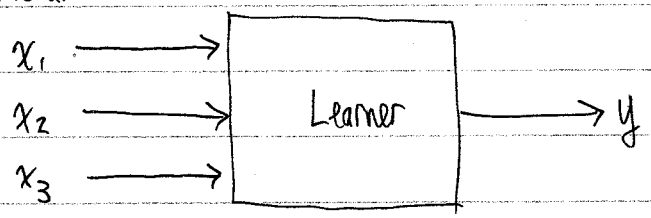
- This is illustrative of some of ICA's role: It is in understanding the nature of the data in order to develop more specialized/optimized analysis on it than ICA. It helps find structure.
- PCA & ICA work remarkably well but other techniques exist. (Also, PCA is slow, and ICA is slower.)
  - RCA = random components analysis, a.k.a. randomized projections
    - Basically, it picks random directions & projects onto them. (Usually a lower # of them)
    - It works, remarkably well, if the next step is classification particularly. It may however project into a larger-dimensional space than is needed.
    - It can also work to project into a larger ~~space~~ rather than smaller space.
    - Big advantages: Simple, easy, fast.
    - It's "irritating", like k-means, because it works, but is really dirt-simple.
  - LDA = linear discriminant analysis
    - Finds a projection that discriminates based on the label.
    - Sort of similar to SVM.
    - This is the only one that pays attention to labels.
- PCA is more linear algebra (and only coincidentally about probability).
- ICA is more probability & information theory (and only coincidentally about linear algebra). Also, independent components don't always exist.

Linear algebra is cheaper and easier. Sometimes, it does something that also relates to probability.

2015-04-05

UL-5 CS7641

- Information Theory (From Pushkar)
  - Where does information theory come up in machine learning?
  - Consider:



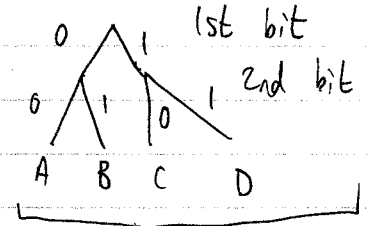
We might ask: How are y & x_i related? Which of x_1, x_2, or x_3 gives the most information about y - and what does information mean here?

- We can view all these input vectors as probability density functions. We might ask...
  - Are these input vectors similar? → Mutual information
  - Has this feature any information? → Entropy
- History: Claude Shannon & Bell Labs
  - It has a background in physics, particularly thermodynamics. See Maxwell's Demon for more on this!
- Consider 10 coin flips by a fair coin (50% chance of heads & tails) and by an unfair coin which is 100% heads.
  The fair coin requires 10 bits to convey outcome (use one bit for heads, other for tails), and other coin needs 0, because result is always the same.
  Less certainty → More information
  - Entropy: What is the min. number of yes/no questions?
- Or, if we have symbols A, B, C, D, each with 25% chance of showing up...

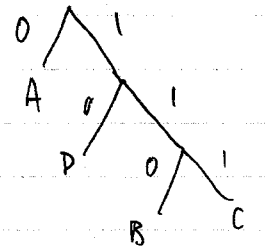
2015-04-05(b)

UL-5 (cont.)

- Then we'd need 2 bits for each. (Say, A=00, B=01, C=10, D=11). But what if A occurred 50% of the time, 12.5% each for B & C, and 25% for D? We could use the same encoding - but could also do better:



Inefficient.
Need 2 questions for each.



Only 1 question for A
2 for D
3 for B & C.

50% · 1 bit
25% · 2 bits
12.5% · 3 bits
12.5% · 3 bits

Σ = 1.75 bits on average.

- This is variable-length encoding.
- We just computed entropy here

as $\sum P(s)\, H(s)$
$= \sum P(s) \log 1/P(s)$
$= -\sum P(s) \log P(s)$

- So, this tells the information in one variable. What about information between two variables?
  - e.g. probability of raining & probability of thunder
  - One way: joint entropy, $H(x,y) = -\sum P(x,y) \log P(x,y)$
  - Another: conditional entropy, $H(y|x) = -\sum P(x,y) \log P(y|x)$
  - If $x \perp\!\!\!\perp y$ (independent):
    $H(Y|X) = H(Y)$ - knowing x doesn't tell anything of y
    $H(x,y) = H(x) + H(y)$ - log simply separates $P(x)P(y)$
- Mutual entropy tells us when variables are independent but next to nothing on their dependence.
- Mutual information: $I(x,y) = H(y) - H(x|y)$ - that is, how much extra does knowing x tell us about y?

$y|x$ = y given x

	2 independent coins	2 dependent coins (assume they always get same as each other)
$P(A)$ & $P(B)$	0.5	0.5
$P(A,B)$	0.25	0.5
$P(A\|B)$	0.5 ($=P(A)$)	$= \frac{P(A,B)}{P(B)} = 1$
$H(A)$	1	1
$H(B)$	1	1
$H(A,B)$	2	1
$H(A\|B)$	1	0
$I(A,B)$	0	1

All pretty trivially evaluated from formulas

- Kullback-Leibler Divergence (or KL Divergence)
  - Measures "distance" between two distributions (not a true distance) → doesn't follow triangle inequality
  - $D(p\|q) = \int p(x) \log \frac{p(x)}{q(x)}$ — Equals 0 iff $p=q$, otherwise >0
  - This comes up because for instance we might be modeling our data as a standard distribution and want to know how closely it fits. Or, it's an approximation to least-squares.

- Notes on Dr. Isbell's "Information Theory" notes:
  - For more info, see: Cover & Thomas, 1991.
  - Variable: object X that can take values from set Ω_X (its domain). Examples: alphabet letters (discrete & finite), $\{0,1\}$, real numbers (continous & infinite).
  - Random variable: variable whose value is unpredictable; a trial is a particular value it takes on at some time, and a collection of trials is a sample.
  - Example: Random variable = coin flip. It takes on values $\{H,T\}$. Each time we flip is a trial. A series of flips is a sample.
  - Every random variable has a probability distribution $P(X)$, possibly unknown, but mapping every value $\in \Omega_X$ to $[0,1]$.
    - e.g. $P(X=H) = P(X=T) = 0.5$ for a fair coin.
      $P(X=H) = 1$, $P(X=T) = 0$ for a coin with two heads.
  - For continous variables we usually use probability density, not distribution.

$$\sum_{x \in \Omega_X} P(x) = 1$$

2016-04-05c, CS7641

Isbell "Information Theory" notes

- Random variables may be <u>dependent</u> on one another, or independent, but grey areas exist in between too.
- <u>Joint distributions</u> formalize this, like $P(X, Y)$, which tells everything about co-occurence of events from X & Y.

  Also, $P(X) = \sum_{y \in \Omega_Y} P(X, Y=y)$ (marginal distribution)

$$P(X,Y) = P(X)P(Y) \iff X \text{ \& } Y \text{ are independent}$$

- <u>Conditional distribution</u>: $P(Y|X) = \frac{P(X,Y)}{P(X)}$

  & Bayes' rule: $P(X|Y) = P(Y|X)\frac{P(X)}{P(Y)}$

- More generally: Joints & conditionals work with any number of variables.

  $P(X) = \sum_{z \in \Omega_Z, y \in \Omega_Y} P(X, Y=y, Z=z)$ for instance; or:

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

- Moments
  - <u>Mean</u> or <u>expected value</u> of random variable:

$$E_X[X] = \sum_{x \in \Omega_X} x\,P(X=x)$$

- or "expectation of X" or simply $E[X]$, to abuse notation.
    - We must know $P(X)$, the distribution, to compute this.
    - If we don't, we can compute <u>sample mean</u>:

$$E[X] = \frac{1}{N} \sum_{x \in \Omega_X} x$$

where N is number of trials in sample.

    - <u>True mean</u> is deterministic function of distr. of X. Sample mean is not.
    - Sample mean depends on random variables being sampled, therefore sample mean itself is a random variable.
      In the limit it approaches true mean ("law of large numbers")
  - Variance: $Var(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2$
    Standard deviation $\sigma(X) = \sqrt{Var(X)}$
  - Mean is the first <u>moment</u>; there are k of them, denoted by $E[X^k]$

Isbell "information theory" notes

- If mean is subtracted first, that gives central moment, $E[(X-E[X])^k]$. Thus, variance = 2nd central moment.
- Also, normalized central moment: $\frac{E[(X-E[X])^k]}{\sigma(X)^k}$ (controls for scale)
- Kurtosis, which we used already, is the 4th normalized central moment & measures peakedness of distribution.

- Entropy
  - often defined: $H(X) = -E[\log P(X)] = -\sum_{x \in \Omega_X} P(X=x) \log P(X=x)$ where log = base 2 & where $0 \log 0 = 0$
  - It captures randomness/uncertainty in variable which in turn measures average message length. (eg. entropy for fair coin = 1 bit; for the always-heads biased coin, 0 bits. For a coin that reads 75% heads, entropy = 0.8113 bits.)
  - As with probabilities we can find conditional & joint entropies.
    - Joint entropy: $H(X,Y) = -E_X[E_Y[\log P(X,Y)]] = -\sum_{x \in \Omega_X, y \in \Omega_Y} P(X=x, Y=y) \log P(X=x / Y=y)$
    - Conditional entropy: $H(Y|X) = -E_X[E_Y[\log P(Y|X)]] = -\sum_{\dots} P(X=x, Y=y) \log P(Y=y|X=x)$
  - Consequences of these:
    $H(Y|X) = H(Y)$ & $H(X,Y) = H(X) + H(Y) \Leftrightarrow$ X & Y are independent
    $H(Y|X) \leq H(Y)$ - knowing more info. can never increase uncertainty
    $H(X,Y) = H(Y|X) + H(X) = H(X|Y) + H(Y)$
    - And generalizes to > 2 variables
  - Continous/differential entropy, h, lacks some of the properties of H. h can be negative, and even if $-\infty$ can still have uncertainty.

- Mutual Information
  - Small $H(Y|X)$ may mean that X tells us a lot about Y! Or, that $H(Y)$ is just small anyway.
  - Mutual information gets around this:
    $I(X,Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$
    $= H(X) + H(Y) - H(X,Y)$
    $= I(Y,X)$
  - It is a "measure of the reduction in randomness of a variable given knowledge of another variable"

2016-04-06, CS7641

Isbell's "Information Theory" notes

- Note that mutual information is a special case of KL divergence!

$$D(p||q) = \int p(x) \log \frac{p(x)}{q(x)}$$ (always non-negative, always positive for $p \neq q$)

- It measures difference between two distributions & is also called relative entropy. It is not symmetric.
- Mutual information is also:

$$D(P(X,Y) \,||\, P(X)P(Y)) = \int P(X,Y) \log \frac{P(X,Y)}{P(X)\,P(Y)}$$

- That is the difference between joint probability, and the product of individual probabilities. (Recall that these are equal only for independent X & Y).

- Notes on: "An introduction to information theory & entropy" (Tom Carter)
  - How would we measure the complexity of something? One way is on how surprising/unexpected an observation/event is → information theory.
  - He uses both 'frequentist' & 'observer relative' versions of probability.
  - What information do we get for seeing an event with probability p?
  - Information:
    - Non-negative, $I(p) \geq 0$
    - If probability 1, no information - $I(1) = 0$
    - If two independent events: $I(p_1 * p_2) = I(p_1) + I(p_2)$
  - From these we derive: $I(p) = -\log_b(p) = \log_b(1/p)$
    - typically $b = 2$ for bits but other units appear too.
  - Now suppose we have n symbols $\{a_1, a_2, \ldots a_n\}$ that some source emits with probabilities $\{p_1, p_2, \ldots p_n\}$ respectively, and symbols are emitted independently. What average amount of information does a symbol carry?
    - Symbol a_i gives us $\log(1/p_i)$. We also have a chance p_i of seeing a_i. Thus, take weighted average:

$$I/N = \sum_{i=1}^{n} p_i \log(1/p_i) \quad \text{for N observations.}$$

Intro. to info. theory & entropy

- Thus, Shannon defined <u>information</u> by probabilities of events alone.
- Entropy $H(P) = \sum_{i=1}^{N} p_i \log(1/p_i)$ - Same definition as I/N.

or, $\int P(x) \log(1/P(x))\, dx$ for continuous distribution

- Or: $H(P) = \langle I(p) \rangle$ where $\langle F \rangle = \sum_{i=1}^{n} F_i p_i$ (expected value)

→ Entropy is just <u>expected value</u> of information of distribution.

- Gibbs inequality, $\ln(x) \leq x - 1$ for $x > 0$.
Now take distributions P & Q, $\{p_1, p_2, \ldots p_n\}$ & $\{q_1, q_2, \ldots q_n\}$, where $p_i, q_i > 0$ & $\sum p_i = \sum q_i = 1$.

$$\sum_{i=1}^{n} p_i \ln\left(\frac{q_i}{p_i}\right) \leq 0$$

- If you actually work through math and use the Gibbs inequality.
- See p29. Short version: This tells us that only an <u>even</u> probability distribution (uniform?) maximizes entropy.
And: $0 \leq H(P) \leq \log(n)$. $H(P) = 0$ if one $p_i = 1$ and rest are zero. $H(P) = \log(n)$ if all $p_i = \frac{1}{n}$.

- One point to get: Information & entropy depend on the underlying probability distribution!
- I ~~stopp~~ stop here but these slides also cover:
  - channel capacity & coding
  - generalized dimensions
    - incl. Hausdorff dimension & fractals
    - D_0 = Hausdorff dimension
    - D_1 = information dimension
    - D_2 = correlation dimension
  - analog signalling systems (we covered only discrete)
- Note that: $KL(P;Q) = \left\langle \log\left(\frac{p(x)}{q(x)}\right) \right\rangle_p$