CECS 451 Lectures

Darin Goldstein

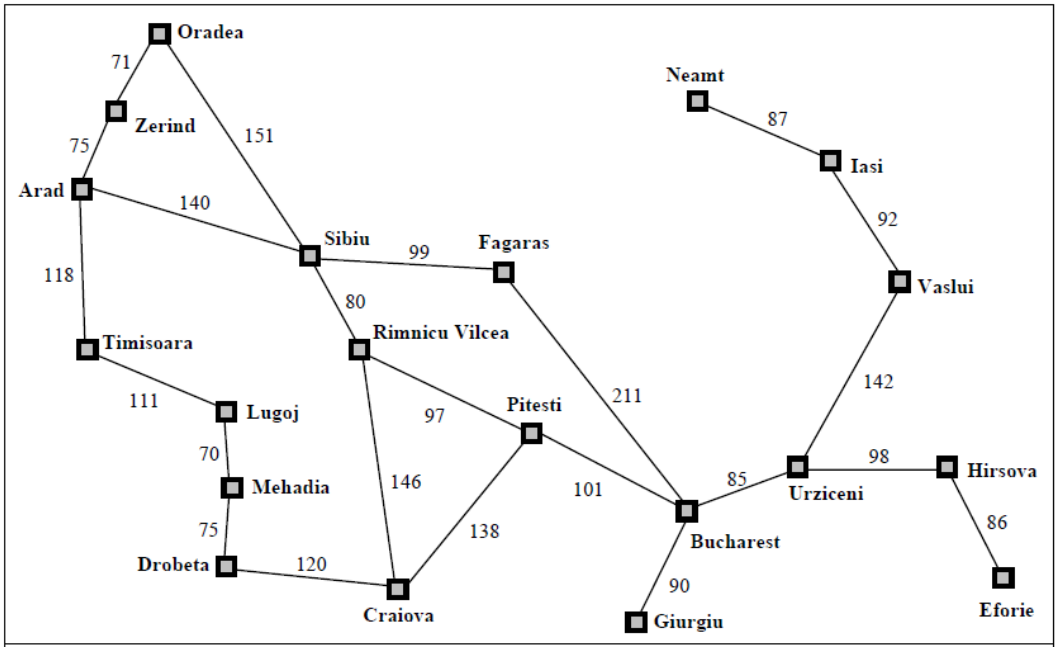
1 A*

Introduce $f'(n) = g'(n) + h'(n)$: $g'(n)$ is the minimum cost to get from the root to node n and $h'(n)$ measures the minimum cost to get from node n to the goal. If we knew f' for every node n, then it would be trivial to find a goal state quickly: given your current state, locate the child that has the same f' value that you do and move to it. In fact, if we know the exact value for h', it is trivial to construct an algorithm as well: at the root, we know the exact value for f'; thus, continue moving to children with this same value.

Unfortunately, it is almost never easy to find g' nor h' so we have to deal with approximations.

Let $g(n)$ be the function that measures approximately the distance from the root to node n. In other words, $g(n)$ measures the smallest cost value to node n from the root that we have discovered thus far. At all times, $g'(n) \leq g(n)$. Let $h(n)$ be a heuristic that approximates the distance from n to the goal state.

The following is a graph of Romania. Amazingly, the cities in Romania all start with different letters of the English language.



Using the map of Romania given above, the following is a list of the straight-line distances from each city to Bucharest.

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Note that this straight-line distance is a reasonable estimate of the remaining driving distance from each city to Bucharest. We will refer to a heuristic that estimates the remaining cost from the current node n to the goal as $h(n)$.

Let Algorithm A be the algorithm that does a best-first search based on the value of $f(n) = g(n) + h(n)$. The search terminates as soon as a goal state is found. Note that the way that Algorithm A is defined, it does not always find the cheapest path to the goal state. (One possible way that the search may not find the optimal path: Consider what happens if there is a single node on the optimal path that vastly overestimates the value of h. The search will never want to expand that node.)

Now let us assume that we can bound the value of h so that $\forall n, h(n) \leq h'(n)$. Then the heuristic is called **admissible**.

1.1 The Algorithm

Consider the following algorithm (assuming that $h(n)$ is an admissible heuristic): Initialize the root node to have $f(root) = g(root) + h(root) = h(root)$ and initialize an empty min-heap. Continue doing the following steps until a goal node is popped off the min-heap: Pop the node n with the smallest f value off of the min-heap. Determine all of n's children and compute their respective $f(n)$ values. If a child of n has already been discovered previously with a smaller or same f value, then discard the newer child with the larger f value. (i.e. If the g estimation was from a path of lesser or equal length to the one we are currently considering, discard the current one.) Insert all remaining children into the min-heap and continue.

1.1.1 Example

Use A^* to find the shortest path from Sibiu to Bucharest.

1.2 Theoretical results for A^*

The following are the interesting theoretical results for A^*.

Definition: A search algorithm is **complete** if, assuming that there exists a path from the root to a goal state, the algorithm is guaranteed to eventually find one before terminating.

Claim: A^* is complete.

Proof: Is it possible for an infinite loop to occur? Clearly not. □

Claim: A^* always finds an optimal path to a goal node.

Proof: By the definition of the algorithm, we can assume that A^* terminates with the goal node. We claim that the path found from the root to the goal node is a shortest path. Assume not; then the path found is strictly suboptimal and there exists a path from the root to the goal node with strictly smaller length; call this latter path P.

Is it possible for all nodes along P to have been popped from the stack? No, if this were so, then the f score on the goal would have been lower. Let n be the first unpopped node along the path P that was not popped from the stack. Because P is a shortest path and all nodes up until n were popped from the stack, we must have that $g(n) = g'(n)$. But then note that because h is admissible, we have the following sequence:

$$f(n) = g(n) + h(n) = g'(n) + h(n) \leq g'(n) + h'(n)$$

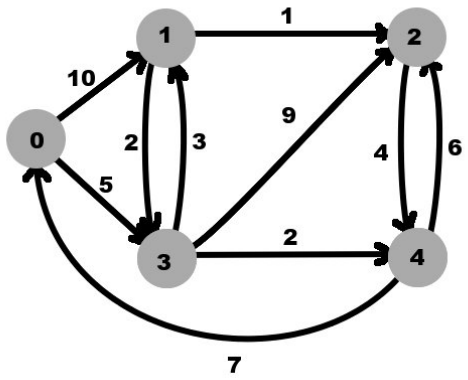
But the quantity on the right is the length of P which we have assumed to be strictly shorter than the f score on the goal node. This implies that n should have been popped before the goal node but wasn't. □

Claim: There is no other complete, optimal algorithm that "expands" fewer nodes with the given heuristic f. A^* is therefore the most "efficient" algorithm for finding an optimal path, given f.

Proof: If algorithm B terminates with a given path to the goal and there exists an unexpanded node with a lower f value, then the algorithm cannot possibly rule out that a shorter path might exist: Replace the current graph with a graph that has an extra edge that makes the heuristic true; the operation of algorithm B will still find the same path to the goal and, this time, it will be incorrect. □

1.3 Dijkstra's algorithm

Consider Dijkstra's algorithm. Find the shortest distance in the following graph from 0 to 4.



Interesting fact: Dijkstra's algorithm is A^* with $h(n) = 0$.

2 Consistency

Definition: Let h be a heuristic such that $\forall n, h(n) \leq c(n, n') + h(n')$ for all children n' of n where $c(n, n')$ is the minimum cost from getting from state n to n'. Let h evaluate to 0 at all goal nodes. Then h is called **consistent**.

Claim: Any consistent heuristic is also admissible.

Proof: Let h be consistent. If n is a node such that cannot reach a goal state, then any h value is admissible. Thus, we restrict our attention to nodes n such that the distance from n to the goal state is strictly finite.

We will prove the claim by induction on number of steps away from the goal state the currently considered node n is along a shortest path. Clearly, if n is a goal state then $h(n) = h'(n) = 0$ (and is thus admissible). Now assume that the heuristic h is admissible on all nodes that are within k steps of the goal node along a shortest path. Let n be a node that is exactly $k+1$ steps away from the goal node along a shortest path. Consider a child of n, say n', that lies along

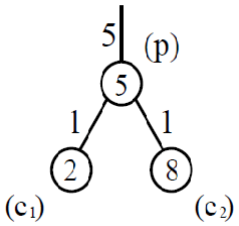
a shortest path to n's closest goal node. Then we have the following (because $h(n') \leq h'(n')$ via induction):

$$h(n) \leq c(n, n') + h(n') \leq c(n, n') + h'(n') = h'(n)$$

□

2.1 Inconsistent heuristics

Consider the following situation from the paper by Felner et al. from 2011. Assume that the unseen node on top is the start node. Let the values inside the nodes be the heuristic values and the values along the edges be the cost of moving from one node to the next.



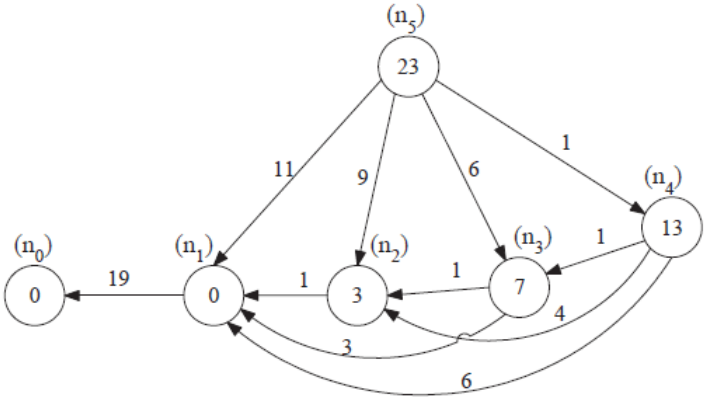
Claim: This heuristic can be made admissible but not consistent.

Proof: To show admissibility, it is enough to show that the actual cost to the goal state is greater than the estimated cost. Let $d(c_1, goal) = 4$ and $d(c_2, goal) = 8$ for admissibility. For consistency, notice that

$$5 = h(p) > d(p, c_1) + h(c_1) = 1 + 2 = 3$$

2.2 Theory

It was proved in 1977 by Martelli that A^* can take $\Omega(2^n)$ steps to find the goal node on an n-node graph if the heuristic is allowed to be inconsistent. This is an example of one of Martelli's graphs.

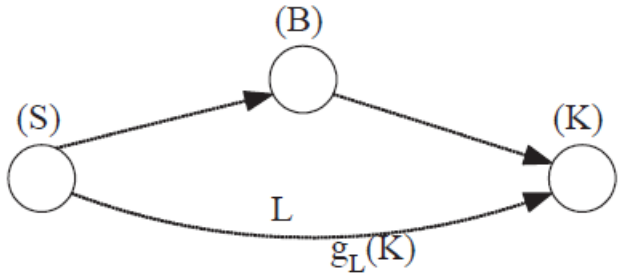


Claim: If A^* takes exponential time to run, then there must exist some edge that is exponentially large. (This usually means that the graph was concocted to be pathological.)

Proof: If A^* expands N nodes but performs M expansions total, then the average number of times a node is re-expanded is $\frac{M-N}{N}$. So at least one node must have been re-expanded that many times. However, we never add a node back to the heap if its f value remains the same or increases. Thus there must exist at least one node with f value at least $\frac{M-N}{N}$. If A^* takes exponential time, then that implies that there must exist a node with exponentially large f value at some point during the algorithm. That implies that either the g or h value of the node is exponentially large. If it is the g value, then there exists some edge with exponentially large edge weight somewhere on the path from the root to the node. If it's the h value, then because h is admissible, there must exist some edge with exponentially large weight somewhere on the path from the root to the node. Either way, the graph contains an exponentially large edge weight. □

Claim: If A^* expands N nodes but performs M expansions total, then the shortest path from the root to a goal node is at least $\frac{M-N}{N}$.

Proof: By the above discussion, there must exist a node that was re-expanded $\frac{M-N}{N}$ times. In the figure, let K be a node that was expanded at least $\frac{M-N}{N}$ times, and let S be the start node. Assume that L is the path that first led to K's expansion, and let B be the node that caused K's last expansion. Let f_L and g_L denote the f and g values along the lower path and f_U and g_U denote the corresponding values along the upper path.



Now we have that because $f_U(B) = g_U(B) + h(B) \Rightarrow$

$$h(B) = f_U(B) - g_U(B)$$

Because K is first expanded via L, we must have that $f_U(B) \geq f_L(K)$ (otherwise, B would have been expanded first and L would never have had the chance to expand K). Thus $f_U(B) \geq f_L(K) \Rightarrow$

$$\geq f_L(K) - g_U(B)$$

$f_L(K) = g_L(K) + h(K) \Rightarrow$

$$= g_L(K) + h(K) - g_U(B)$$

$g_U(B) \leq g_U(K) \Rightarrow$

$$\geq g_L(K) - g_U(K) + h(K)$$

$h(K) \geq 0 \Rightarrow$

$$\geq g_L(K) - g_U(K)$$

$g_L(K) - g_U(K) \geq \frac{M-N}{N} \Rightarrow$

$$\geq \frac{M-N}{N}$$

Thus there must be some node B that was expanded via A^* with h value at least $\frac{M-N}{N}$. Now note that B was expanded before the goal node.

$$\text{optimal path length from root to goal} \geq f(goal) \geq f(B) \geq h(B) \geq \frac{M-N}{N}$$

□

Claim: Using A^*, if the optimal path length is $\leq \lambda$ and $M > N$ then $M \leq N(\lambda + 1)$.

Proof: From above $\frac{M-N}{N} \leq \lambda$. □

Claim: If the edge weights of a given search graph are $O(1)$, then $M = O(N^2)$ even if the heuristic is inconsistent.

Proof: Notice that the optimal path length can only be of length $O(N)$ if the edge weights are $O(1)$. But by the above claim, we must have that $\frac{M-N}{N} = O(N) \Rightarrow M = O(N^2)$. □

The A^* algorithm is guaranteed to return an optimal solution only if an admissible heuristic is used. There is no requirement that the heuristic be consistent. It is usually assumed that admissible heuristics are consistent. In their popular AI textbook Artificial Intelligence: A Modern Approach, Russell and Norvig write that "one has to work quite hard to concoct heuristics that are admissible but not consistent". Many researchers work under the assumption that "almost all admissible heuristics are consistent". The term "inconsistent heuristic" has, at times, been portrayed negatively, as something that should be avoided. Part of this is historical: early research discovered that inconsistency can lead to poor performance for A^*. However, the issue of inconsistent heuristics has never been fully investigated or re-considered after the invention of IDA^* (iterative deepening A^*). These perceptions about inconsistent heuristics are wrong.

Many domains have a number of heuristics available. When using only one heuristic the search may enter a region with"bad" (low) estimation values (a heuristic depression). With a single fixed heuristic, the search is forced to traverse a (possibly large) portion of that region before being able to escape from it. A well-known solution to this problem is to consult a number of heuristics and take their maximum value. When the search is in a region of low values for one heuristic, it may be in a region of high values for another. There is a tradeoff for doing this, as each heuristic calculation increases the time it takes to compute $h(n)$. Additional heuristic consultations provide diminishing returns in terms of the reduction in the number of node expansions, so it is not always recommended to use them all. Given a number of heuristics one could select which heuristic to use randomly. Only a single heuristic will be consulted at each node, and no additional time overhead is needed over a fixed heuristic. Random selection between heuristics introduces more diversity to the values obtained in a search than using a single fixed selection. The random selection of a heuristic will produce inconsistent values if there is no or little correlation between the heuristics. Furthermore, a random selection of heuristics might produce inconsistent h-values even if all the heuristics are themselves consistent.

3 Optimizing functions blindly

Most of the functions that we deal with in this class will be, for the most part, differentiable (though not necessarily pleasant). In practice though, functions may not be quite as nice. This lecture will give you some very basic tools for finding the minimum of a function over a finite state space where the function is too ugly to differentiate.

The goal is to search for the value of x that minimizes the value of an objective function $f(x)$. Note that the function f can be defined over essentially any reasonable state space: the only requirement that we set up is that any value of x must have well-defined "neighbors". Examples of $f(x)$ include the standard mathematical functions in multiple dimensions that you already know about, but extend all the way out to some truly strange state spaces. Below, x may be

referred to as a **state**.

3.1 Simulated annealing

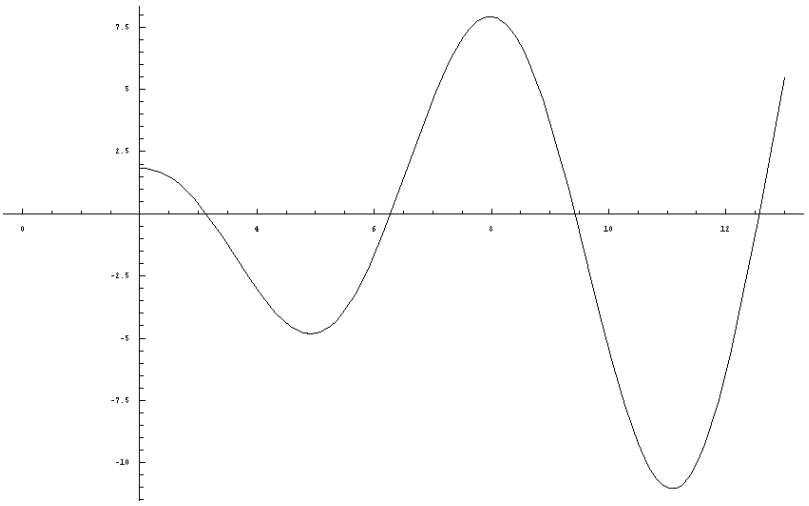
You need to find the global minimum of a function f(x) that you know nothing about. The function f(x) is assumed to be programmed into a black box that you can enter inputs to and receive outputs from, but not look inside.

In order of higher understanding, simulated annealing can be understood in the following ways:

1. Think of the search space as a box with a strangely formed bottom. Place several small balls in the box and shake it up. If all goes well, the balls will all fall into the deepest well in the box, the global minimum of the function.

2. Again, the search space should be though of as a box with grooves in the bottom. This time we release bouncing balls inside the box with a certain amount of energy: the higher the energy we give the balls, the higher the balls bounce. Gradually lower the energy of the balls in the box. As the energy of the balls gets lower, it will become more and more difficult for the balls to bounce out of the deepest valleys. If we lower the energy slowly enough, the balls should eventually come to rest where we want them to, the global minimum of the function.

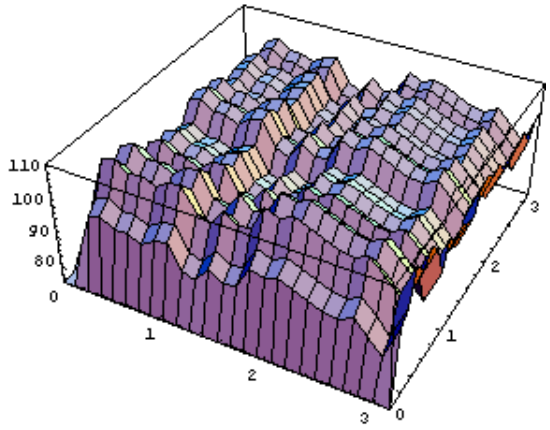
3. Instead of thinking of f(x) inside a box, we are now thinking of it as the energy function of particles of a gas. By the 2nd law of thermodynamics, any system will eventually tend to its ground (lowest) energy state. Assume that you release a gas into a box. If the temperature in the box is high, the particles will interact and not tend to remain in any given energy state for very long. As we start to lower the temperature in the box, the number of states that the particle can jump to is cut down. The activation energy is the energy that a particle requires to jump from one energy state (local minima) to another. In the graph, it corresponds to the difference between the energy of the current local minimum and the local maximum between the other energy state. For example, if the average kinetic energy of this gas is over 20, then a gas particle could occupy either of the energy states of this function (local minima) because the activation energy needed to make the transition from one state to the other is less than 20. On the other hand, if the temperature is slowly lowered so that the average kinetic energy is 15, then particles have the energy to make the jump from the left to the right state but not back again.



We should not lower the temperature of the gas too quickly or the particles will be trapped prematurely in local minima energy states. On the other hand, if we take too long to lower the temperature, the algorithm will run forever. Determining the "right" rate of temperature lowering is the main art of simulated annealing.

1. Via magic, choose the number of particles N, the initial temperate $T > 0$, and the cooling parameter $0 < \alpha < 1$.
2. Place the N particles randomly around the state space.
3. While the previous equilibrium configuration is not equal to the next equilibrium configuration, continue performing the following steps.
   (a) While the number of particle transitioning to lower energy states is greater than the number of particles transitioning to higher states (i.e. until equilibrium is reached), continue performing the following steps.
      i. For each particle in the space located at x, choose a randomly selected neighbor x^*. If $f(x^*) < f(x)$, then move the particle at x to x^*. If $f(x^*) \geq f(x)$, then move the particle at x to x^* with probability $e^{\frac{f(x)-f(x^*)}{T}}$.
   (b) Replace the value of T with αT.
4. Choose the particle with the lowest value of $f(x)$.

The following function was generated by choosing one hundred random r_i^x values and one hundred random r_i^y values all uniformly chosen between 0 and 30 and then computing $\sum_{i=1}^{100} \sin^2(r_i^x x) + \sin^2(r_i^y y)$ where $x, y \in [0, \pi]$. The function looks like this:



With 40 particles, a starting temperature of 400, and cooling parameter $\frac{3}{4}$, simulated annealing found a minimum of 81.6959. In contrast, the built in Mathematica function NMinimize found a minimum of 84.4502. The true minimum is 0, of course, when $x, y = 0$, but the average expected value of the function is 99.17. The final temperature was 0.0000955668.

Simulated annealing has been put into fairly effective use in practice: designing molecules, determining the optimal design for airline wings (and other optimization problems), robotic movements, and task scheduling.

3.2 Genetic algorithms

Note that maximization of functions is the same problem as minimization of functions because maximizing $f(x)$ is the same problem as minimizing $-f(x)$. In this section, we will be looking to maximize $f(x)$ rather than minimize.

Consider the game Mastermind. There is some special code consisting of a string of 15 English characters that someone has encoded into a box. For example, the code might be DJPOCVNWGHIQWPV. The object of the game will be to figure out this code. However, you cannot open the box and look inside. The box is set to self-destruct if anyone looks in. You can feed in guesses to the box and the box will output the following information: number of correct letters in correct positions AND number of correct letters in incorrect positions. With this information, you can continue to make better and better guesses as to what the code is...

Say, for example, that we have the following 15 character code: JFMCPWMVSJSNBOT. You make the following guess to the black box: XQMWRBWGKLTMTOB. The black box will output the following information:

number of correct characters in correct positions: 2

number of correct characters in incorrect positions: 4

The two correct characters in the correct positions are the M in the third position and the O in the 14th position. The four correct characters in incorrect positions are W,B,M, and T.

Using only our guesses and the information we get from them, how can we possibly get to a special 15 character code? Notice that there is no possible way to guess every code. There are $26^{15} = 1677259342285725925376$ possible codes to try. You need to somehow use your past intelligent guesses to come up with future ones that are better...

We want to make a fitness function that awards points to those guesses that are closest to the special code. For any guess, let the number of correct characters in correct positions be a and the number of correct characters in incorrect positions be b. Then the fitness score of that guess will equal $2^a + b - 1$. (There are many other fitness scores possible. This probably isn't even the best one.) Notice that the maximum score that any guess can have is 32,767: 15 correct characters in correct positions.

Assume that there are three individuals whose fitness scores are 3, 7, and 2 respectively. Assume that the environment will only support 1 individual. In this case, the individual that survives is chosen in direct proportion to his fitness score. The probability that the individual with score 3 will survive, for example, is $3/12 = 1/4$. In this way, the strongest individual survives to the next generation with the highest probability, but not with certainty.

1. Choose values for the following parameters: Maximum population (50 individuals), Mutant population (usually fairly small, 15 individuals), Crossover population (usually relatively large, 75 individuals)

2. The initial generation will consist of (e.g. 50) individuals chosen at random. No duplicate genetic codes are allowed.

3. For as many generations as necessary, continue to do the following steps.

   (a) Starting with the current population, select a number of individuals at random that will have mutant children. Create a mutation population (e.g. 15 individuals) consisting of copies of the selected individuals with a single mutation (e.g. either (a) a character gets changed at random or (b) two characters get switched at random).

   (b) Starting with the current population, create a crossover population (e.g. 75 individuals). Each crossover individual will be the merging of two parents chosen randomly in proportion to their fitness score. To merge two parents into a child, choose a value for each character of the child's that is the character for one of the parents chosen at random.

4. Choose 50 members for the next generation in proportion to their fitness score from the combined population of the previous generation, the mutations, and the crossovers. Duplicate genes are not allowed.

One should be careful when performing a genetic search: it is difficult to tell when the search has reached a local maximum. A local maximum is a point that is larger than every point in its immediate surroundings but not the largest

possible in the entire search space. There are different methods that researchers use to avoid falling into the traps of local maxima, but without some knowledge of the search space, it is an intractably difficult problem to overcome.

Some genetic algorithms work surprisingly well. Research on certain specific problems has produced some good results. However, unless you know something about the state space you are searching beforehand, it is difficult to get a genetic algorithm to work. The choices of what possible mutations there are, mutant population size, how two genetic individuals mate, the offspring population size, etc. make constructing a genetic algorithm more an art than a science.

4 Optimizing functions intelligently

4.1 Gradient ascent

Given a curve $f(x_1, x_2, \ldots, x_n)$ in n-dimensional space that is smooth, we wish to be able to find the equation of a tangent hyperplane at some point $(x'_1, \ldots, x'_n)$. This is clearly

$$f(x'_1, \ldots, x'_n) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}|_{(x'_1,\ldots,x'_n)}(x_i - x'_i)$$

because this is the only linear function that is tangent to the curve along every coordinate.

Notice that the slope of the curve can be discerned directly from the tangent plane and is clearly

$$\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$$

Thus, the direction of steepest ascent is in the same direction as the gradient.

The main idea behind gradient ascent is, given your current position, locate a better one that is close to you and move there. The algorithm is as follows: Choose a random starting position $p = (x_1, \ldots, x_n)$, a step size $\alpha > 0$, and a threshold $\epsilon > 0$. Let the function in question be $f(x_1, \ldots x_n)$. While the change between the positions is greater than ϵ, change the position of the particle by $\alpha \nabla f|_p$.

The movie shows the idea behind gradient ascent with the function defined by $f(x, y) = \sin(1/2x^2 - 1/4y^2 + 3)\cos(2x + 1 - e^y)$. We start the initial position of the particle at just slightly below the origin: $(-.1, -.1)$. Notice that the particle does make it to the top of the nearest local maximum but does not hit the global maximum.

4.2 Lagrange multipliers (introduction)

Find the point on the line $y = mx + b$ that is closest to the point (x', y').

We wish to minimize $(x - x')^2 + (y - y')^2$ subject to the constraint that $y = mx + b$. Form the Lagrangian as follows:

$$f(x, y, \alpha) = (x - x')^2 + (y - y')^2 + \alpha(y - (mx + b))$$

Now take the derivative with respect to x, y, α and set them all equal to 0.

$$\frac{\partial f}{\partial x} = 2(x - x') - \alpha m = 0$$

$$\frac{\partial f}{\partial y} = 2(y - y') + \alpha = 0$$

$$\frac{\partial f}{\partial \alpha} = y - (mx + b) = 0$$

These imply that

$$y = \frac{b + m(x' + my')}{m^2 + 1}, x = \frac{(x' + my') - bm}{m^2 + 1}$$

$$\alpha = -\frac{2((mx' + b) - y')}{m^2 + 1}$$

Note that the Lagrangian function itself is unbounded in both the positive and negative direction (e.g. let $y \to \infty$ and all other variables go to 0; let $\alpha \to sign(b)\infty$ and all other variables go to 0). The moral of this story is that we are not looking to find the global maximum of the Lagrangian; we are looking to find the local maxima so that the constraints are all satisfied. Gradient ascent/descent is perfect for this in case simple derivatives are not available.

5 Perceptrons

5.1 Introduction

In this section, we will assume that we are using a very simple kind of classification machine called a perceptron. In this model, we will assume that we have a vector of parameters $\theta = (\theta_0, \theta_1, \ldots, \theta_n)$ that are linearly related to the inputs to our model $x = (1, x_1, \ldots, x_n)^T$.

> **Whenever we introduce a model, the parameter θ will represent a row vector and the x inputs will be a column.**

Note that we always assume that the first input x_0 is 1; this component allows us to adjust by a constant. Let the function s (for "step") be $s(x) = 1$ if $x \geq 0$ and $s(x) = -1$ otherwise. Then, given a sequence of inputs x, our hypothesis is given by $s(\theta x)$.

For example, if $\theta = (0, 1, -1)$ and $x = (1, 6, 4)^T$, then the result is +1. On the other hand, for the same perceptron $x = (1, 4, 6)^T$ yields a result of -1.

There are many functions that can be performed by a perceptron (assuming 0/1 inputs):

- AND: Let $\theta = (-1.5, 1, 1)$.
- OR: Let $\theta = (-0.5, 1, 1)$.

- NOT: Let $\theta = (0.5, -1)$.

However, a single perceptron cannot handle XOR. We'll show this is in two ways.

Claim: Perceptrons cannot deal with XOR.

Proof: Assume that the perceptron is given by $\theta = (\theta_0, \theta_1, \theta_2)$. Then we must have the following four inequalities satisfied for XOR:

1. $\theta_0 + \theta_1 + \theta_2 < 0$
2. $\theta_0 + \theta_1 - \theta_2 \geq 0$
3. $\theta_0 - \theta_1 + \theta_2 \geq 0$
4. $\theta_0 - \theta_1 - \theta_2 < 0$

Add the first and last equations and the second and third equations to get an immediate contradiction. □

There is another way to see that perceptrons will never be able to handle XOR. Any perceptron simply splits the space in half and chooses one of the halves to be positive and the other negative: Note that if θ consists of constants and the x's are variables (except for the first which is always 1), the equation θx describes a hyperplane. Data sets that can be perfectly classified by a "plane-splitter" are called *linearly separable*. Is the data set corresponding to the XOR function linearly separable?

Notice that the negative points are $(1, 1)$ and $(0, 0)$ and the positive points are $(1, 0)$ and $(0, 1)$. Try separating these two sets with a single line in the plane... it can't be done.

5.2 Perceptron confidence

For the purposes of classification, we simply take the sign of the result and make the classification. However, notice that the term itself $|\theta x^{(i)}|$ yields a measure of how confident we are in our classification. If this value is small, then we are not as confident in our classification as if it were large; the reason for this is that a small positive (resp. negative) value is much closer to the negative (resp. positive) range than a large positive (resp. negative) value. If the value of θ gets adjusted slightly, then our classification of $x^{(i)}$ might change.

5.3 Learning with perceptrons

Assume that we are being fed example inputs $x^{(1)}, \ldots, x^{(m)} \in \mathcal{R}^n$ and a "correct answer" $y^{(1)}, \ldots, y^{(m)} \in \{-1, 1\}$ for each. The way we evaluate and learn is as follows: Initialize the model θ to 0. For each i, evaluate $s(\theta x^{(i)})$. We now compare $s(\theta x^{(i)})$ to $y^{(i)}$. If the values are the same, then we continue with the next example. If the values are different, we modify our parameter vector θ via the mapping $\theta += y^{(i)}(x^{(i)})^T$. This is called the perceptron learning algorithm.

The intuition behind this is as follows: If the classification we get is the correct one, we don't want to change anything. If the classification we get is incorrect, we want to move the θ parameter in some direction. The way to improve θ the best is by moving it in the direction of the sample that it got wrong: If the sample classifies $x^{(i)}$ as negative when it should be positive, then θ needs to move in the positive $x^{(i)}$ direction and vice versa for negative. Mathematically, if we assume that we classify $x^{(i)}$ incorrectly as negative with θ as the parameter vector, then

$$(\theta + (x^{(i)})^T)x^{(i)} = \theta x^{(i)} + ||x^{(i)}||^2$$

is more likely to be positive than it was before ($\theta x^{(i)}$) because the squared norm of any vector is guaranteed to be positive. The same reasoning holds for the other case.

5.4 Cauchy-Schwarz inequality

Claim: $\forall a, b \in \mathcal{R}^n, ||a||||b|| \geq a \cdot b$

Proof: Let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$. We have

$$0 \leq \frac{1}{2} \sum_{i,j=1}^{n} (a_i b_j - a_j b_i)^2 =$$

$$\sum_{i=1}^{n} a_i^2 \sum_{j=1}^{n} b_j^2 - (\sum_{i=1}^{n} a_i b_i)^2 =$$

$$||a||^2 ||b||^2 - (a \cdot b)^2$$

□

6 Online bounds for perceptron learning

6.1 Perceptron learning algorithm bounds for online learning

Theorem(Block, Novikoff - separately 1962): Suppose that

$$\exists D, \forall i, ||x^{(i)}|| \leq D$$

(i.e. there is a uniform bound on how large the example vectors are allowed to be) and that there exists some vector u such that $||u|| = 1$ and that

$$\forall i, y^{(i)}(ux^{(i)}) \geq \gamma$$

(i.e. that there exists some reasonable assignment to the parameters that makes our predictions confident by amount at least γ). Then the total number of mistakes that the perceptron algorithm makes on this sequence is at most $(D/\gamma)^2$.

It is important to notice that this bound is independent of both m and n and is therefore immune to the curse of dimensionality!

Proof: Fix the u that is referred to in the statement of the theorem.

Define $\theta^{(k)}$ to be the value of the parameters of the model during the kth mistake. So, for example, $\theta^{(1)} = 0$ because we initialized θ to 0.

Lemma 1:

$$\forall k \geq 0, (\theta^{(k+1)})u^T \geq k\gamma$$

Proof of Lemma 1: We will prove this by induction. It is trivially true for $k = 0$. Assume that it is true for k. Then because of the definition of the learning rule (assuming that the kth mistake was made on input i)

$$(\theta^{(k+1)})u^T = \left[\theta^{(k)} + y^{(i)}(x^{(i)})^T\right] u^T$$

By the assumption that $\forall i, y^{(i)}(ux^{(i)}) \geq \gamma \Rightarrow y^{(i)}(x^{(i)})^T u^T \geq \gamma$, we have that

$$\left[\theta^{(k)} + y^{(i)}(x^{(i)})^T\right] u^T \geq \theta^{(k)}u^T + \gamma$$

By the inductive assumption

$$\theta^{(k)}u^T + \gamma \geq (k-1)\gamma + \gamma = k\gamma$$

□

Lemma 2:

$$\forall k \geq 0, ||\theta^{(k+1)}||^2 \leq kD^2$$

Proof of Lemma 2: We will also prove this by induction. It is trivially true for $k = 0$. Assume that it is true for k. By the definition of the learning algorithm (assuming that the kth mistake was made on input i)

$$||\theta^{(k+1)}||^2 = ||\theta^{(k)} + y^{(i)}(x^{(i)})^T||^2$$

Expanding the right hand side and noting that $y^{(i)} \in \{-1, 1\}$

$$||\theta^{(k)} + y^{(i)}(x^{(i)})^T||^2 = ||\theta^{(k)}||^2 + ||x^{(i)}||^2 + 2y^{(i)}\theta^{(k)}x^{(i)}$$

Note that if the ith input is a mistake, then we must have that $y^{(i)}\theta^{(k)}x^{(i)} \leq 0 \Rightarrow$

$$||\theta^{(k)}||^2 + ||x^{(i)}||^2 + 2y^{(i)}\theta^{(k)}x^{(i)} \leq ||\theta^{(k)}||^2 + ||x^{(i)}||^2$$

By the definition of D

$$||\theta^{(k)}||^2 + ||x^{(i)}||^2 \leq ||\theta^{(k)}||^2 + D^2$$

By the inductive hypothesis

$$||\theta^{(k)}||^2 + D^2 \leq (k-1)D^2 + D^2 = kD^2$$

□

By Lemma 2, we have that $\sqrt{k}D \geq ||\theta^{(k+1)}||$. By the Cauchy-Schwarz inequality, we have that $||\theta^{(k+1)}|| = ||\theta^{(k+1)}||||u|| \geq \theta^{(k+1)}u^T$. Combining these two, we get $\sqrt{k}D \geq \theta^{(k+1)}u^T$. Taking this inequality together with Lemma 1 yields the result. □

7 Neural networks

In computer science, if an object with inputs and outputs is "orderly," then there exists a computer program that is able to reproduce the action of the given object. The "complexity" of an object with inputs and outputs refers to the length of the shortest computer program that could reproduce the action of the given object. Ask yourself the following question: What is the most complex yet orderly arrangement of matter known to man?

The average brain has 100 billion neurons with between 1,000 and 10,000 synapses per neuron. Each neural synapse can perform about 200 calculations per second. It only takes up about 2,200 cubic centimeters in your head. Your brain uses about 20% of the oxygen you breathe and 15-20% of your blood gets pumped to your brain. If you don't get blood to your brain for 8-10 seconds, you lose consciousness; 40-110 seconds without blood to the brain, you lose your reflexes.

Assume each neuron has approximately 5,000 synapses. With 100 billion neurons in the average person's brain, this works out to 500,000,000,000,000 total synapses. On an ordinary double-spaced typewritten page, there are roughly 2,000 characters. In a 375 page textbook, there are therefore about 750,000 characters. The United States Library of Congress, as of 2015, holds about 37,000,000 books total. There are more synapses in your brain than there would be characters on every page in every book in over 18 Libraries of Congress (a small city filled exclusively with books).

Each synapse can perform 200 calculations per second. That is MUCH slower than even your PC at home. Why are you so much smarter than your PC at home?[1] Answer: Even though computers are orderly, they aren't complex.

Neural networks can be used to perform simple, straightforward tasks. For example... Handwriting recognition, Speech recognition, Automatic spelling correction, Speech translation, Lip Reading, Gesture Recognition, Medical Diagnosis, Fraud Detection, Predicting Stock Prices

Neural nets in games: Neural networks are generally used when you want something done in real time that would otherwise take a great deal of calculation. Obstacle avoidance: It is possible for the computer to precisely calculate where and when it will run into a wall, but... Aiming weapons: Again, if the computer knows where an opponent is going... Adapting to changing strategies...

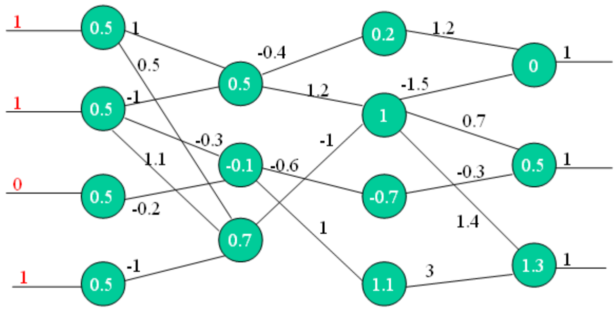
7.1 Idealized neural nets

For every neuron, a threshold is defined. If the synapse has fired then the weight on the edge counts towards the input of the neuron that the synapse points to. To determine whether a neuron has fired, sum up the values of the ON synapses

[1] Interesting fact: In 2002, IBM was awarded 200 million dollars by the government to build a supercomputer that could perform approximately the same number of calculations per second that the human brain could... Why is IBM's computer still not going to be as smart as you are?

that lead in to the neuron. If that value meets or exceeds the threshold, then the neuron fires and all of its output synapses turn ON.

Determine what the outputs of this network are if the inputs are as shown.

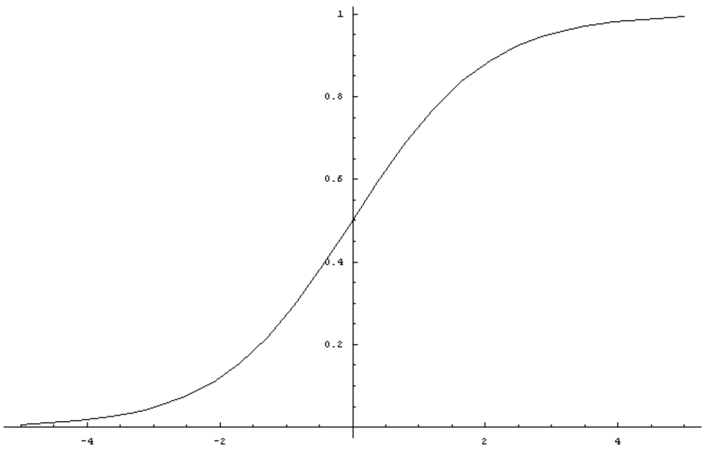


What inputs might lead to the output 0,1,0 (top to bottom)?

Note that it is theoretically possible to assume that the threshold of any given neuron is 0. If the threshold of a neuron is x, then replace the neuron with a neuron with threshold 0 that has an input synapse coming in with weight $-x$ that is always on.

7.2 Threshold functions

The **threshold function** is a function that measures how ON either a neuron or a synapse really is. (Thus far, we have been assuming that things are either 100 percent ON or OFF.) The threshold function that we will use is $t(x) = \frac{1}{1+e^{-x}}$.



Notice that the derivative of the threshold function has a simple form.

$$\frac{d}{dx}t(x) = \frac{d}{dx}\frac{1}{1+e^{-x}} = \frac{e^{-x}}{(1+e^{-x})^2} = t(x)(1-t(x))$$

What would the outputs be of the above network if the inputs are as shown assuming we use $t(x)$ as the threshold function rather than the step function? From top to bottom, the first column yields

$$0.622, 0.622, 0.378, 0.622$$

The second column yields

$$0.378, 0.46, 0.419$$

The third column yields

$$0.413, 0.276, 0.604, 0.345$$

The outputs are

$$0.52, 0.38, 0.53$$

Notice that these values are fairly different from the values we calculated above.

7.3 Definitions and notation

- Assume that we are given a neural network with L levels.
- Let N_i be the set of neurons on level i where i can run from 1 to L.
- For any neuron n, let $I(n)$ be the set of input edges to n and let $O(n)$ be the set of output edges from n.
- Let S be the set of sample inputs to the network.
- If s is a sample input and n is a neuron, let $f_n(s)$ be the value that determines whether the neuron FIRES or not. Recall that we are no longer using the step function... We are now using the function $t(x)$.
- If s is a sample input and n is a neuron on level L, let $w_n(s)$ be the value that we want $f_n(s)$ to be.
- If there is a connection $n' \to n$ between two neurons, let $x_{n',n}$ be the weight of the synapse.

Note that our definitions imply that for any neuron n,

$$f_n = t(\sum_{n' \in I(n)} x_{n',n} f_{n'})$$

Let the performance function be

$$P = -\sum_{s \in S} \sum_{n \in N_L} (f_n(s) - w_n(s))^2$$

What is the maximum value of this function?

8 Teaching neural networks

Chain rule: Assume that there is a function f that is a function of $y_1, \ldots, y_n$. Assume that we wish to find the derivative of f with respect to x, some other variable.

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{n} \frac{\partial f}{\partial y_i} \frac{\partial y_i}{\partial x}$$

The Gradient: Assume that you are given a function f that is a function of $y_1, ..., y_n$. It is a fact that the direction of steepest ascent of the function f is in the direction of the gradient of f.

$$\nabla f(y_1, y_2, \ldots, y_n) = (\frac{\partial f}{\partial y_1}, \ldots, \frac{\partial f}{\partial y_n})$$

8.1 Definitions and notation

- Assume that we are given a neural network with L levels.
- Let N_i be the set of neurons on level i where i can run from 1 to L.
- For any neuron n, let $I(n)$ be the set of input edges to n and let $O(n)$ be the set of output edges from n.
- Let S be the set of sample inputs to the network.
- If s is a sample input and n is a neuron, let $f_n(s)$ be the value that determines whether the neuron FIRES or not. Recall that we are no longer using the step function... We are now using the function $t(x)$.
- If s is a sample input and n is a neuron on level L, let $w_n(s)$ be the value that we want $f_n(s)$ to be.
- If there is a connection $n' \to n$ between two neurons, let $x_{n',n}$ be the weight of the synapse.

8.2 The Algorithm

The backpropagation algorithm is as follows:

1. Compute the value of

   $$\frac{\partial P}{\partial f_{n_2}} = 2 \sum_{s \in S} (w_{n_2}(s) - f_{n_2}(s))$$

   for nodes such that $n_2 \in N_L$.

2. For nodes such that $n_2 \notin N_L$, compute the values

   $$\frac{\partial P}{\partial f_{n_2}} = \sum_{n' \in O(n_2)} f_{n'}(1 - f_{n'}) x_{n_2,n'} \frac{\partial P}{\partial f_{n'}}$$

   starting from the next to last neural layer and working backwards.

3. Compute the weight changes for each edge weight x_{n_1,n_2} using the following formula.

$$\Delta x_{n_1,n_2} = r f_{n_2}(1 - f_{n_2}) f_{n_1} \frac{\partial P}{\partial f_{n_2}}$$

4. Adjust the weights of the edges accordingly.

$$x_{n_1,n_2} \mapsto x_{n_1,n_2} + \Delta x_{n_1,n_2}$$

8.3 Derivation

Note that our definitions imply that for any neuron n,

$$f_n = t(\sum_{n' \in I(n)} x_{n',n} f_{n'})$$

Let the performance function be

$$P = -\sum_{s \in S} \sum_{n \in N_L} (f_n(s) - w_n(s))^2$$

What is the maximum value of this function? To maximize the performance of the neural net, we need to maximize the performance function and find the place where the gradient is 0. Note that the quantities that we are allowed to vary are the $x_{n',n}$ values and we therefore need to determine $\frac{\partial P}{\partial x_{n',n}}$ for each valid synapse.

P depends on x_{n_1,n_2} only via f_{n_2}. Thus, via the chain rule, we have that

$$\frac{\partial P}{\partial x_{n_1,n_2}} = \frac{\partial P}{\partial f_{n_2}} \frac{\partial f_{n_2}}{\partial x_{n_1,n_2}}$$

Working on $\frac{\partial f_{n_2}}{\partial x_{n_1,n_2}}$...

$$\frac{\partial f_{n_2}}{\partial x_{n_1,n_2}} = \frac{\partial f_{n_2}}{\partial \sum_{n' \in I(n_2)} x_{n',n_2} f_{n'}} \frac{\partial \sum_{n' \in I(n_2)} x_{n',n_2} f_{n'}}{\partial x_{n_1,n_2}}$$

$$\frac{\partial \sum_{n' \in I(n_2)} x_{n',n_2} f_{n'}}{\partial x_{n_1,n_2}} = f_{n_1} \Rightarrow$$

$$\frac{\partial f_{n_2}}{\partial x_{n_1,n_2}} = f_{n_1} \frac{\partial f_{n_2}}{\partial \sum_{n' \in I(n_2)} x_{n',n_2} f_{n'}}$$

$$f_{n_2} = t(\sum_{n' \in I(n_2)} x_{n',n_2} f_{n'}) \Rightarrow$$

$$\frac{\partial f_{n_2}}{\partial \sum_{n' \in I(n_2)} x_{n',n_2} f_{n'}} = t(x)(1 - t(x))|_{x = \sum_{n' \in I(n_2)} x_{n',n_2} f_{n'}} = f_{n_2}(1 - f_{n_2})$$

Thus, we have

$$\frac{\partial f_{n_2}}{\partial x_{n_1,n_2}} = f_{n_1} f_{n_2}(1 - f_{n_2})$$

Working on $\frac{\partial P}{\partial f_{n_2}}$...

There are two cases to consider. Either $n_2 \in L$ or not.

1. $n_2 \in L$:

$$P = -\sum_{s \in S} \sum_{n \in N_L} (f_n(s) - w_n(s))^2 \Rightarrow$$

$$\frac{\partial P}{\partial f_{n_2}} = 2 \sum_{s \in S} (w_{n_2}(s) - f_{n_2}(s))$$

2. $n_2 \notin L$:

$$\frac{\partial P}{\partial f_{n_2}} = \sum_{n' \in O(n_2)} \frac{\partial P}{\partial f_{n'}} \frac{\partial f_{n'}}{\partial f_{n_2}}$$

We work on the second term.

$$\frac{\partial f_{n'}}{\partial f_{n_2}} = \frac{\partial t(\sum_{n'' \in I(n')} x_{n'',n'} f_{n''})}{\partial f_{n_2}}$$

$$= \frac{\partial t(\sum_{n'' \in I(n')} x_{n'',n'} f_{n''})}{\partial \sum_{n'' \in I(n')} x_{n'',n'} f_{n''}} \frac{\partial \sum_{n'' \in I(n')} x_{n'',n'} f_{n''}}{\partial f_{n_2}}$$

This expression has been encountered before...

$$= f_{n'}(1 - f_{n'}) x_{n_2,n'}$$

And finally...

$$\frac{\partial P}{\partial f_{n_2}} = \sum_{n' \in O(n_2)} f_{n'}(1 - f_{n'}) x_{n_2,n'} \frac{\partial P}{\partial f_{n'}}$$

Thus, we pick a parameter r, the rate at which we want the neural net to learn. If we pick r too large, the neural net may "overlearn" and treat each example as more important than it should. On the other hand, if we pick r too small, the neural net may take too long to learn the data.

If we think of the edge weights as a vector x, then the optimal move for x is in the direction of the gradient of the performance function P with respect to x. The move we will make will therefore be $r\nabla P$.

Thus, for each edge weight x_{n_1,n_2}, we calculate the change in the edge weight via $\Delta x_{n_1,n_2} = r\frac{\partial P}{\partial x_{n_1,n_2}}$.

9 Deriving the information function

Shannon information: If you are sending a character over a line, the amount of information you get from the character should depend on the frequency that the character is sent. Intuitively, the information function should measure the surprise you feel at receiving the given character. Let $c(x)$ be a character that is sent with probability x. The information function I should have the following properties:

1. I should depend on x.
2. $I(x) + I(y) = I(xy)$
3. $I'(x)$ is defined everywhere on $(0, 1]$ from the left hand side (because of $x = 1$)
4. $I'(1)$ is negative and finite.

Let x be any character that is sent with probability strictly less than 1. First, note that by rule 2, we must have that $I(1) = 0$. Then

$$\begin{aligned}
I'(x) &= \lim_{h\to 0^+} \frac{I(x) - I(x-h)}{h} \\
&= \lim_{h\to 0^+} \frac{I(x) - I(x) - I(1-\frac{h}{x})}{h} \\
&= \lim_{h\to 0^+} -\frac{I(1-\frac{h}{x})}{h} \\
&= \lim_{h\to 0^+} \frac{1}{x} I'(1-\frac{h}{x}) \\
&= \frac{I'(1)}{x} \\
\Rightarrow \quad I(x) &= I'(1)\log_2 x + C
\end{aligned}$$

Letting $x = 1$, we get that $C = 0$ so that the amount of information contained in a character that appears with probability x is $I(x) = I'(1)\log_2 x$.

Note that the value of $I'(1)$ changes the base of the log depending on how you measure the information. You can measure information in bits, trits, digits, etc. From now on, we will assume that the base of the log is 2 so that we are always measuring in bits.

The *entropy* of a sequence of bits is the average quantity of information that you get per character in the sequence. We denote it by

$$H(\{x_1, x_2, \ldots, x_n\}) = \sum_{i=1}^{n} x_i I(x_i)$$

Another way of looking at the entropy is this: The entropy measures the smallest theoretical code that can be created for a given set of character frequencies (in

the sense that any other uniquely decipherable code must have at least this many bits per codeword on average). What are the best and worst cases for sending information over a line using only two characters? We use the entropy to compare how good a code is: the closer the average codeword size is to the entropy, the better the code.

A *uniquely decipherable* code is one that has no ambiguity: a given set of characters always has a unique decoding. A *prefix code* is defined to be a code in which no codeword is also a prefix of some other codeword. A prefix code is decodeable on the fly. Is this a prefix code: {0,10,11}? Is this a prefix code: {0,1,10,11}?

Most and least efficient: What are the minimum and maximum entropy of n-character uniquely decipherable codes? Clearly, the minimum entropy is 0. This entropy arises if one of the characters appears with probability 1 and the rest 0. This corresponds to the "perfect" model, one we always predict with absolute certainty. To find the maximum entropy, note that we are looking to maximize the function $g(p) = -\sum_{i=1}^{n} p_i \log_2 p_i$. Introduce the Lagrangian $f(p, \alpha) = g(p) + \alpha(\sum_{i=1}^{n} p_i - 1) = 0$. So we get that for every i,

$$\frac{\partial f}{\partial p_i} = -\log_2 p_i - \frac{1}{\ln 2} + \alpha = 0 \Rightarrow$$

$$\forall i, j, p_i = p_j \Rightarrow p_i = \frac{1}{n} \Rightarrow H(p) = \log_2 n$$

Notice that this corresponds to the number of bits necessary to distinguish n distinct objects. What does this analyze say about, for example, the standard encoding of characters in an average novel or the efficiency of the English language as a whole? □

In order to find the *optimal* values for paramters in a model, our goal will be to *minimize* the amount of information that our model yields given our measurement. This seems counterintuitive at first. Why would we want to minimize the amount of information coming out of our model? Recall that information measures the surprise that you feel when you discover the measurement... our basic goal then is to minimize our total surprise given our measurements. Another way of looking at it is that if our model is very bad, we will probably be very surprised by normal measurements.

10 Categorization models

The data set in auto-mpg.csv is a data set taken fro the UCI repository from 1983 where each record is the make a particular brand of car (where the names have been removed). The goal is to set up a prediction model that, given some or all of the attributes, predicts the mpg of the car. Note that (assuming there are no contradictions or "mistakes" in the data) it is always possible to create a decision tree that *perfectly* categorizes the data! Simply create branches until each leaf holds exactly 1 record. This is referred to as **overfitting** the data.

Our goal will be to create a model where there is significant information gain in the model with every decision made by the model. Because mpg is a continuous variable, we will separate the mpg levels into three regimes: strictly less than 25 mpg, greater than or equal to 25 but less than 32, and strictly greater than 32. Note that out of 391 records, exactly 225 fall into the first category (a), 100 in the second category (b), and 66 fall into the third category (c). Therefore the entropy of the "empty" model (in bits) is 1.39513.

Our ultimate goal will be to minimize the information in the model, as usual; however, we want to only ask question that yield a significant decrease in the overall information of the current model. For example, one question we might ask of the data is "Is the acceleration of the car strictly less than 16 or not?" If we ask this question, then the data yields the following information.

- YES answer: 152 (a), 45 (b), 29 (c)
- NO answer: 73 (a), 55 (b), 37 (c)

This means that the entropy of new model is

$$(152 + 45 + 29)/391 H(152/226, 45/226, 29/226) +$$

$$(73 + 55 + 37)/391 H(73/165, 55/165, 37/165) = 1.35683$$

Given this example, we will start by attempting to answer the following question: Given some data, some threshold height h, and a positive real number r, does there exist a tree with height h such that the average information from the model is less than or equal to r? This problem (properly phrased, which we did not) is NP-complete (proved by Laurent Hyafil, Ronald L. Rivest).

This is a decrease in overall surprise; however, it only amount to a roughly 3 percent decrease. Is there a question that yields a better overall information decrease in the model? Let's try "Was the car made strictly before 1978?"

- YES answer: 176 (a), 56 (b), 9 (c)
- NO answer: 49 (a), 43 (b), 58 (c)

This yields an entropy of

$$(176 + 56 + 9)/391 H(176/241, 56/241, 9/241) +$$

$$(49 + 43 + 58)/391 H(49/150, 43/150, 58/150) = 1.21871$$

This is a decrease in overall surprise of over 12%. We will call this significant enough to keep.

We can now focus on each of the nodes separately. On the YES answer, let's ask the question: "Does the car have 4 or fewer cylinders, 6 cylinders, or 8 or greater cylinders?" Note that in the YES answer node, there are only 241 examples as opposed to 391.

- 4 or fewer cylinders: 38 (a), 56 (b), 9 (c)

- 6 cylinders: 53 (a), 0 (b), 0 (c)
- 8 or more cylinders: 85 (a), 0 (b), 0 (c)

To compute the expected information coming out of this model, we need to determine the probability that we wind up in each node and then calculate the average entropy.

$$(49 + 43 + 58)/391 H(49/150, 43/150, 58/150) +$$

$$[(176 + 56 + 9)/391][(38 + 56 + 9)/241] H(38/103, 56/103, 9/103) +$$

$$[(176 + 56 + 9)/391][(53 + 0 + 0)/241] H(53/53, 0, 0) +$$

$$[(176 + 56 + 9)/391][(85 + 0 + 0)/241] H(85/85, 0, 0) = 0.954016$$

This is a hefty decrease. Notice especially that the nodes in which everything was exactly the same contributed nothing to the overall sum; they were classified perfectly. Also note that we could have merged the two nodes with perfect classifications into a single node and not lost anything. (Though we will not go over this in this class, there exist algorithms where merging is in fact one of the allowed operations due to the fact that sometimes our guess for which question to ask may need to be tweaked.)

There are examples of useless splits. For example, if there is a 2/2 split, then splitting this node into two nodes of 1/1 and 1/1 is completely useless *and* wastes a step on the way to categorization. The general greedy algorithm for categorization is as follows (assuming a maximum depth of k): Starting with the empty model node, recursively find the question that minimizes the information in the model and then split the node according to this question assuming the split does not cause a node with depth lower than k. This is a greedy method and, interestingly, does not always yield the decision tree with the smallest height.

There are various heuristics for choosing the greedy question, alternatives to information gain as the measure of "goodness" of a given question, and methods for pruning trees. We won't go into that kind of depth though.

10.1 When does the greedy method fail?

We will assume for the sake of this example that categorization algorithms will only ask questions that classify based on a single field.

Let the number of items in the data set be 64. Let there be 2 primary binary fields and another secondary field. Assume that we split the data set perfectly equally among the 2 primary fields so that after 2 questions (assuming that we only ask questions about the primary fields), all items are classified perfectly. Imagine, for example, taking the first (resp. second, third, fourth) 16 items and giving them classification 0 (resp. 1, 2, 3). Let the secondary field be 4-ary and can thus have any one of the values in $\{0, 1, 2, 3\}$. Let the secondary field be assigned in such a way that if the secondary field is measured to be i, then it is

highly likely to find lots of items with classification i (for $0 \leq i \leq 3$); specifically, 13 items of classification i and 1 each of the other types.

What question will the greedy algorithm ask first? Will it attempt to classify based on the primary fields or the secondary?

Let's consider the information loss if we ask a single YES/NO question from either primary/binary field. Clearly, the data will be split perfectly down the middle. Thus, the entropy will go from (assuming all logarithms are base 2)

$$\sum_{i=0}^{3} \frac{16}{64}(-\log \frac{16}{64}) = 2 \text{ bits}$$

to

$$\frac{1}{2}(2)(-\frac{16}{32}\log \frac{16}{32}) + \frac{1}{2}(2)(-\frac{16}{32}\log \frac{16}{32}) = 1 \text{ bit}$$

So the entropy decreased by 1 bit, as expected.

Now let's ask what the information loss would be if we ask the question that classifies on the secondary field. As we calculated above, the entropy starts at 2 bits. The final *expected* entropy is

$$\frac{1}{4}(4)(-\frac{13}{16}\log \frac{13}{16} - 3\frac{1}{16}\log \frac{1}{16}) = 0.993393 \text{ bits}$$

Thus, the greedy algorithm would choose to expand on the secondary field first because the entropy is lower, but no matter what primary field you expand next, you cannot possibly reach perfect classification.

10.2 A different kind of categorization: k-means algorithm

This is your first unsupervised learning algorithm. In the clustering problem, you are given a training set of data $\{x^{(1)}, \ldots, x^{(m)}\}$ and you want to group the data into k "clusters." We will let $x^{(i)} \in \mathcal{R}^n$ but notice that no values for what we would usually call $y^{(i)}$ are given; we simply have data that needs to be dealt with and you get no additional help (hence, unsupervised).

The k-means clustering algorithm as follows.

1. Initialize *cluster centroids* $\mu_1, \ldots, \mu_k \in \mathcal{R}^n$ randomly.
2. Perform the following steps until convergence (i.e. until the μ_i values do not change by more than some predetermined $\epsilon > 0$).
   (a) For each i, let $c^{(i)}$ be the value of j that minimizes $||x^{(i)} - \mu_j||$. In other words, $c^{(i)}$ holds the index of the currently closest mean.
   (b) For each j, let $\mu_j = \frac{\sum_{i=1}^{m} \delta_{c^{(i)},j} x^{(i)}}{\sum_{i=1}^{m} \delta_{c^{(i)},j}}$. In other words, let each μ_j be redefined to be the centroid of the points that were closest to it.

11 Nearest neighbor classification

One of the simplest classification algorithms is as follows: Given a set of feature vectors $x^{(i)} \in \mathcal{R}^n$ with associated classifications $y^{(i)}$, the goal is to choose the best classification for some new feature vector x. To do this, simply search for the k nearest $x^{(i)}$ points to x and take a vote to determine the winner classification (where k is some constant chosen in advance). If $k = 1$, then this is simply nearest neighbor classification.

Notice that I never specified the domain set for the $y^{(i)} \in Y$. This is because there are many interesting domains.

- The simplest case is when $|Y|$ is finite; this is a simple classification problem.
- If $Y = \mathcal{R}^m$ for some m, then we have a regression problem.
- If Y is the power set of some set Z, then we have a multilabel problem. For example, we might wish to classify the set of all newspaper articles with the power set of labels {SPORTS,POLITICS,BUSINESS, ... }.
- If $X = A^*$ and $Y = B^*$ where A and B are sets of alphabet symbols, then we have a sequence labeling problem. For example, A^* might be the set of all English sentences, and B^* might be the set of part-of-speech sequences such as NOUN VERB ADJECTIVE.

A quick practical word: This method suffers from the "curse of dimensionality:" The time necessary for computations increases quickly with d, the dimension of the feature vector, and, perhaps more importantly, the accuracy of the method tends to deteriorate with high d as well. In a high-dimensional space, all points in a reasonably sized data set tend to be far away from each other, so nearest neighbors may not be meaningfully similar. (Consider placing a reasonable number of uniformly randomly selected points inside the unit cube in d dimensions. If the number of points is reasonably sized, then most of them will appear near the edge of the cube. Thinking about it in 3 dimensions, if you choose three random numbers between 0 and 1, the chances are roughly one half that one of them will be above 0.9 or below 0.1.) If examples are represented using many features, then every pair of examples will likely disagree on many features, so it will be rather arbitrary which examples are closest to each other.

11.1 Bayes error rate

The Bayes error rate for a classification problem is the minimum achievable error rate, i.e. the error rate of the best possible classifier. This error rate will be nonzero if the classes overlap. For example, if there are two Gaussians on the one-dimensional line, given a new data point on the line, it is impossible to say with 100% certainty which of the two Gaussians generated the point, no matter where it is.

The mathematical definition of the Bayes error rate is the average over the space of all examples of the minimum error probability for each example. The optimal prediction for any example x is the label that has the highest probability given x. The error probability is then one minus the probability of this label. Denote by $x \mapsto y$ the statement that x's classficiation is y. Let $P^B(e|x) = 1 - \max_y P(x \mapsto y)$ be the probability that the Bayesian classifier makes an error given that the data point to be classified is x. Then the average Bayesian error is given by

$$BE = \int_x P(x)P^B(e|x)dx$$

where the maximum over y ranges over all possible labels for x. Note that the Bayes error rate depends on knowledge of the process that is creating the data, and so in that very specific sense, the Bayes error rate is the lowest error rate possible for any reasonable classification scheme.

11.2 Bounds on the nearest neighbor classification algorithm

Theorem (Cover and Hart, 1967): Denote by x and its nearest neighbor by $nn(x)$. Let n be the size of the training set. For any input set X where $\forall i, \lim_{n\to\infty} P(x^{(i)} \mapsto y^{(i)}) = P(nn(x^{(i)}) \mapsto y^{(i)})$, the error rate of the nearest neighbor classifier approaches a number less than or equal to twice the Bayes error rate.

Proof: Assume that the number of possible classificiations is m. We will label these classifications with a label in $\{1, 2, \ldots, m\}$.

Now the probability of an error, given x and its associated $nn(x)$, is the probability that x and $nn(x)$ are of different classes.

$$P(e|x, nn(x)) = 1 - \sum_{i=1}^{m} P(x \mapsto i)P(nn(x) \mapsto i)$$

As $n \to \infty$, this expression approaches

$$P(e|x) = 1 - \sum_{i=1}^{m} P(x \mapsto i)^2$$

(Note that $P(e|x)$ is actually a random variable in x.) Thus, the asymptotic overall error of the nearest neighbor algorithm in the limit as $n \to \infty$ is therefore (This is exactly the same formula that we used for the Bayesian error rate without the Bayes restriction.)

$$\int_x P(e|x)P(x)dx = \int_x [1 - \sum_{i=1}^{m} P(x \mapsto i)^2]P(x)dx$$

$P(e|x)$ is clearly maximized when $\sum_{i=1}^{m} P(x \mapsto i)^2$ is minimized. Keeping in mind the definition of $P^B(e|x)$, we are looking to minimize $\sum_{i=1}^{m} P(x \mapsto i)^2$ subject to $P^B(e|x) + \max_{1\le j\le m} P(x \mapsto j) = 1$ and $\sum_{k=1}^{m} P(x \mapsto k) = 1$.

For the purposes of ease of keeping track of what are and are not variables, let $z_i \equiv P(x \mapsto i)$. Then the problem becomes minimize $\sum_{i=1}^{m} z_i^2$ subject to $P^B(e|x) + \max_{1 \leq j \leq m} z_j = 1$ and $\sum_{k=1}^{m} z_k = 1$.

$$f(z, \alpha, \beta) = \sum_{i=1}^{m} z_i^2 + \alpha(P^B(e|x) + \max_{1 \leq j \leq m} z_j - 1) + \beta(\sum_{k=1}^{m} z_k - 1)$$

Let the j corresponding to $\max_{1 \leq j \leq m} z_j$ be j^*. Taking appropriate partial derivatives...

$$\frac{\partial f}{\partial z_{i \neq j^*}} = 2z_i + \beta = 0 \Rightarrow \forall i, j \neq j^*, z_i = z_j$$

$$\frac{\partial f}{\partial z_{j^*}} = 2z_{j^*} + \alpha + \beta = 0$$

$$\frac{\partial f}{\partial \alpha} = P^B(e|x) + z_{j^*} - 1 = 0 \Rightarrow z_{j^*} = 1 - P^B(e|x)$$

$$\frac{\partial f}{\partial \beta} = \sum_{k=1}^{m} z_k - 1 = 0 \Rightarrow (1 - P^B(e|x)) + \sum_{k \neq j^*} z_k = 1 \Rightarrow$$

$$P^B(e|x) = (m-1) z_{i \neq j^*} \Rightarrow z_{i \neq j^*} = \frac{P^B(e|x)}{m-1}$$

Plugging these results in, we find that

$$1 - \sum_{i=1}^{m} z_i^2 \leq 2P^B(e|x) - \frac{m}{m-1} P^B(e|x)^2$$

This implies that the asymptotic overall error of the nearest neighbor algorithm in the limit as $n \to \infty$ is therefore upper bounded by

$$\int_x [2P^B(e|x) - \frac{m}{m-1} P^B(e|x)^2] P(x) dx$$

Now notice that the variance of any random variable is always greater than or equal to 0. Let $P^B(e|x)$ be the random variable in question. Computing the variance and setting it greater than or equal to 0...

$$\int_x [P^B(e|x) - \int_x P^B(e|x) P(x) dx]^2 P(x) dx \geq 0 \Rightarrow$$

$$\int_x P^B(e|x)^2 P(x) dx \geq (\int_x P^B(e|x) P(x) dx)^2$$

Recalling what we have proved thus far...

$$\int_x P(e|x) P(x) dx \leq \int_x [2P^B(e|x) - \frac{m}{m-1} P^B(e|x)^2] P(x) dx \Rightarrow$$

Using the conclusion above...

$$\int_x P(e|x)P(x)dx \leq \int_x 2P^B(e|x)P(x)dx - \frac{m}{m-1}(\int_x P^B(e|x)P(x)dx)^2$$

And finally...

$$E[P(e|x)] \leq E[P^B(e|x)](2 - \frac{m}{m-1}E[P^B(e|x)])$$

Because the expectation of a probability is at least 0, we are done. □

12 Filtering

12.1 Model setup and assumptions

Assume that your setup is in a sequence of states $s_0, \ldots, s_n$ and that you make a sequence of measurements $m_0, \ldots, m_n$.

The Markov assumption is $P(s_t|s_{0:t-1}) = P(s_t|s_{t-1})$.

The sensor model assumption is $P(m_t|s_{0:t}, m_{0:t-1}) = P(m_t|s_t)$.

Bayes Rule: $P(A|B) = \frac{P(A \cap B)}{P(B)}$. Notice that if $P(B)$ is a constant, then $P(A|B) \sim P(B|A)P(A)$.

Because of Bayes Rule, we get the following that may come in useful below.

$$P(A) = \sum_B P(A \cap B) = \sum_B P(A|B)P(B)$$

For two row vectors $s = (s_1, \ldots, s_n)$ and $t = (t_1, \ldots, t_n)$,

$$s * t \equiv (s_1 t_1, s_2 t_2, \ldots, s_n t_n)$$

12.2 Filtering

Goal: to compute the distribution $P(s_{t+k}|m_{1:t})$.

Assume that you are given that a mole is one one of 3 squares on a pie. The Markov matrix is

$$\begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \end{pmatrix}$$

He has an equal probability of starting on any given square. You can perform a measurement of any given third of the pie, but the measurement is only 75% accurate. Assume that you measure the following sequence of spots and get the following results (assuming the thirds are labelled 1,2,3) where m_t^p represents the measurement at time t on pie third p:

1. $m_1^2 = YES$
2. $m_2^1 = NO$

3. $m_3^1 = NO$
4. $m_4^3 = NO$
5. $m_5^2 = YES$
6. $m_6^1 = NO$

What is the distribution of his location during time $t = 10$?

We get the following because the measurements are constant

$$P(s_{t+1}|m_{1:t+1}) = P(s_{t+1}|m_{1:t}, m_{t+1}) \sim P(m_{t+1}|s_{t+1}, m_{1:t})P(s_{t+1}|m_{1:t}) =$$

by the sensor model assumption

$$P(m_{t+1}|s_{t+1})P(s_{t+1}|m_{1:t}) =$$

by Bayes Rule

$$P(m_{t+1}|s_{t+1}) \sum_{s_t} P(s_{t+1}|s_t, m_{1:t})P(s_t|m_{1:t}) =$$

by the Markov assumption

$$P(m_{t+1}|s_{t+1}) \sum_{s_t} P(s_{t+1}|s_t)P(s_t|m_{1:t})$$

In each of the following, notice that the "unknown" varying item is s_t: Let the row vector of estimates up to time t, namely $P(s_t|m_{1:t})$, be denoted by $forward_t$. Let the row vector of sensor measurement probabilities, namely $P(m_t|s_t)$, be $sensor_t$. Let the Markov transition matrix be denoted by M. Then this formula states that

$$forward_{t+1} \sim sensor_{t+1} * (forward_t M)$$

So to solve the problem posed above, let $forward_0 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Then we get

1. $sensor_1 = (\frac{1}{4}, \frac{3}{4}, \frac{1}{4})$
2. $sensor_2 = (\frac{1}{4}, \frac{3}{4}, \frac{3}{4})$
3. $sensor_3 = (\frac{1}{4}, \frac{3}{4}, \frac{3}{4})$
4. $sensor_4 = (\frac{3}{4}, \frac{3}{4}, \frac{1}{4})$
5. $sensor_5 = (\frac{1}{4}, \frac{3}{4}, \frac{1}{4})$
6. $sensor_6 = (\frac{1}{4}, \frac{3}{4}, \frac{3}{4})$

Working out the first step in the process, we get that

$$forward_1 = sensor_1 * (forward_0 M) =$$

$$(\frac{1}{4}, \frac{3}{4}, \frac{1}{4}) * \left[(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \end{pmatrix} \right] =$$

$$(\frac{1}{4}, \frac{3}{4}, \frac{1}{4}) * (\frac{1}{3}, \frac{7}{18}, \frac{5}{18}) =$$

$$(\frac{1}{12}, \frac{7}{24}, \frac{5}{72}) \sim$$

$$(\frac{3}{16}, \frac{21}{32}, \frac{5}{32})$$

For the sake of completeness the next results are

$$forward_2 = (0.0581395, 0.505814, 0.436047)$$

$$forward_3 = (0.204918, 0.411202, 0.38388)$$

$$forward_4 = (0.469638, 0.418663, 0.111699)$$

$$forward_5 = (0.0546247, 0.766449, 0.178927)$$

$$forward_6 = (0.0677201, 0.476477, 0.455803)$$

So at this point, you have a pretty good idea that the mole is in one of states 2 or 3. After this point, there are no more measurements, so the sensor vector becomes $(1, 1, 1)$.

$$forward_7 = (0.455803, 0.283385, 0.260812)$$

$$forward_8 = (0.260812, 0.445561, 0.293627)$$

$$forward_9 = (0.293627, 0.396655, 0.309718)$$

$$forward_{10} = (0.309718, 0.394079, 0.296203)$$

Now, it's harder to tell where the mole is because we ran out of measurements...

13 Smoothing

Now assume that we are given the same problem as yesterday with exactly the same assumptions, but we want to know the distribution of the guy's location at time $k = 3$.

Because measurements are constant

$$P(s_k|m_{1:t}) = P(s_k|m_{1:k}, m_{k+1,t}) \sim P(s_k|m_{1:k})P(m_{k+1:t}|s_k, m_{1:k})$$

by the sensor model assumption

$$= P(s_k|m_{1:k})P(m_{k+1:t}|s_k)$$

Note that the first term is merely the filtering term $forward_k$ (which can be solved the same way as before). The second term is new. By Bayes Rule

$$P(m_{k+1:t}|s_k) = \sum_{s_{k+1}} P(m_{k+1:t}|s_k, s_{k+1})P(s_{k+1}|s_k) =$$

by the sensor model assumption

$$\sum_{s_{k+1}} P(m_{k+1:t}|s_{k+1})P(s_{k+1}|s_k) =$$

$$\sum_{s_{k+1}} P(m_{k+1}, m_{k+2:t}|s_{k+1})P(s_{k+1}|s_k) =$$

by Bayes Rule

$$\sum_{s_{k+1}} P(m_{k+1}|s_{k+1}, m_{k+2:t})P(m_{k+2:t}|s_{k+1})P(s_{k+1}|s_k) =$$

by the sensor model assumption

$$\sum_{s_{k+1}} P(m_{k+1}|s_{k+1})P(m_{k+2:t}|s_{k+1})P(s_{k+1}|s_k)$$

In each of the following, notice that the "unknown" varying item is s_i: Let the row vector of values $P(m_{i+1:t}|s_i)$ be called $backward_i$. Let the row vector of sensor measurement probabilities, namely $P(m_i|s_i)$, be $sensor_i$. Let the Markov transition matrix be M. Note first that $backward_t$ is a solid column of all 1 values (because the measurements are empty, the probability of measuring them is 1). The above formula states that

$$backward_i = (backward_{i+1} * sensor_{i+1})M^t$$

The goal is to have i run from $t-1$ downwards. We stop the process when we get to $i = k$.

To compute the final distribution, we use

$$P(s_k|m_{1:t}) \sim forward_k * backward_k$$

From above, we know that $forward_3 = (0.204918, 0.411202, 0.38388)$. We will now compute $backward_3$... Note that $backward_6 = (1, 1, 1)$. So we get that

$$backward_5 = (backward_6 * sensor_6)M^t =$$

$$\left((1,1,1) * (\frac{1}{4}, \frac{3}{4}, \frac{3}{4})\right) \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \end{pmatrix}^t =$$

$$(\frac{1}{4}, \frac{3}{4}, \frac{3}{4}) \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 1 & 0 & 0 \end{pmatrix}^t =$$

$$(\frac{3}{4}, \frac{3}{4}, \frac{1}{4})$$

For completeness, we have the following:

$$backward_4 = (0.395833, 0.3125, 0.1875)$$

$$backward_3 = (0.171875, 0.140625, 0.296875)$$

Finally,

$$forward_3 * backward_3 = (0.204918, 0.411202, 0.38388) * (0.171875, 0.140625, 0.296875) =$$

$$(0.0352203, 0.0578253, .113964)$$

Normalizing this so that the components sum to (roughly) 1:

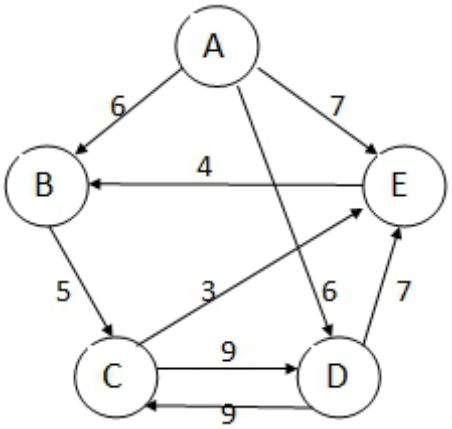
$$(0.170138, 0.279336, 0.550525)$$

The interesting part of this is that after the forward calculation, the most likely place for the mole was in the second slice of the pie, but after factoring in what happened later on, it more than likely that the mole was actually in the third slice.

14 The Viterbi Algorithm I

14.1 APSP

Solve the APSP using min-plus matrix operations. Use the following directed graph as an example:



The adjacency matrix for this graph is as follows:

$$\begin{pmatrix} \infty & 6 & \infty & 6 & 7 \\ \infty & \infty & 5 & \infty & \infty \\ \infty & \infty & \infty & 9 & 3 \\ \infty & \infty & 9 & \infty & 7 \\ \infty & 4 & \infty & \infty & \infty \end{pmatrix}$$

The results of the min-plus multiplication (each time multiplying by the adjacency matrix on the right) are as follows. If we require two steps...

$$\begin{pmatrix} \binom{E}{\infty} & \binom{E}{11} & \binom{B}{11} & \binom{E}{\infty} & \binom{D}{13} \\ \binom{E}{\infty} & \binom{E}{\infty} & \binom{E}{\infty} & \binom{C}{14} & \binom{C}{8} \\ \binom{E}{\infty} & \binom{E}{7} & \binom{D}{18} & \binom{E}{\infty} & \binom{D}{16} \\ \binom{E}{\infty} & \binom{E}{11} & \binom{E}{\infty} & \binom{C}{18} & \binom{C}{12} \\ \binom{E}{\infty} & \binom{E}{\infty} & \binom{B}{9} & \binom{E}{\infty} & \binom{E}{\infty} \end{pmatrix}$$

If we require three steps...

$$\begin{pmatrix} \binom{E}{\infty} & \binom{E}{17} & \binom{B}{16} & \binom{C}{20} & \binom{C}{14} \\ \binom{E}{\infty} & \binom{E}{12} & \binom{D}{23} & \binom{E}{\infty} & \binom{D}{21} \\ \binom{E}{\infty} & \binom{E}{20} & \binom{B}{12} & \binom{C}{27} & \binom{C}{21} \\ \binom{E}{\infty} & \binom{E}{16} & \binom{B}{16} & \binom{E}{\infty} & \binom{D}{25} \\ \binom{E}{\infty} & \binom{E}{\infty} & \binom{E}{\infty} & \binom{C}{18} & \binom{C}{12} \end{pmatrix}$$

If we require four steps...

$$\begin{pmatrix} \binom{E}{\infty} & \binom{E}{18} & \binom{B}{22} & \binom{C}{25} & \binom{C}{19} \\ \binom{E}{\infty} & \binom{E}{25} & \binom{B}{17} & \binom{C}{32} & \binom{C}{26} \\ \binom{E}{\infty} & \binom{E}{25} & \binom{B}{25} & \binom{C}{21} & \binom{C}{15} \\ \binom{E}{\infty} & \binom{E}{29} & \binom{B}{21} & \binom{C}{25} & \binom{C}{19} \\ \binom{E}{\infty} & \binom{E}{16} & \binom{D}{27} & \binom{E}{\infty} & \binom{D}{25} \end{pmatrix}$$

Notice that the shortest path from C to B takes 2 steps, goes through E, and is of length 7. However, the second matrix claims that the path is of length 20, and the final matrix claims that the shortest path is of length 25. Why? Because these matrices *force* you to take more steps than you need. How do we figure out the shortest path from C to B that takes exactly 4 steps? First look in the (C,B) entry of the 4-step matrix. This entry has a $\binom{E}{25}$ in it; this implies that the final step in the path is E. So in the 3-step matrix, we are looking for the shortest path from C to E. This new entry contains a $\binom{C}{21}$: the path now looks like C-?-C-E-B. So the shortest 2-step path from C to C can be determined by looking at the (C,C) entry of the 2-step matrix: $\binom{D}{18}$. So our path is C-D-C-E-B.

What would the solution look like if the cost of each edge decreased by 1 after every step? Obviously, the first step (the adjacency matrix itself) is the same, but the 2-step matrix looks like the following.

$$\begin{pmatrix} \binom{E}{\infty} & \binom{E}{10} & \binom{B}{10} & \binom{E}{\infty} & \binom{D}{12} \\ \binom{E}{\infty} & \binom{E}{\infty} & \binom{E}{\infty} & \binom{C}{13} & \binom{C}{7} \\ \binom{E}{\infty} & \binom{E}{6} & \binom{D}{17} & \binom{E}{\infty} & \binom{D}{15} \\ \binom{E}{\infty} & \binom{E}{10} & \binom{E}{\infty} & \binom{C}{17} & \binom{C}{11} \\ \binom{E}{\infty} & \binom{E}{\infty} & \binom{B}{8} & \binom{E}{\infty} & \binom{E}{\infty} \end{pmatrix}$$

The 3-step matrix is

$$\begin{pmatrix} \binom{E}{\infty} & \binom{E}{14} & \binom{B}{13} & \binom{C}{17} & \binom{C}{11} \\ \binom{E}{\infty} & \binom{E}{9} & \binom{D}{20} & \binom{E}{\infty} & \binom{D}{18} \\ \binom{E}{\infty} & \binom{E}{17} & \binom{B}{9} & \binom{C}{24} & \binom{C}{18} \\ \binom{E}{\infty} & \binom{E}{13} & \binom{B}{13} & \binom{E}{\infty} & \binom{D}{22} \\ \binom{E}{\infty} & \binom{E}{\infty} & \binom{E}{\infty} & \binom{C}{15} & \binom{C}{9} \end{pmatrix}$$

The 4-step matrix is

$$\begin{pmatrix} \binom{E}{\infty} & \binom{E}{12} & \binom{B}{16} & \binom{C}{19} & \binom{C}{13} \\ \binom{E}{\infty} & \binom{E}{19} & \binom{B}{11} & \binom{C}{26} & \binom{C}{20} \\ \binom{E}{\infty} & \binom{E}{19} & \binom{B}{19} & \binom{C}{15} & \binom{C}{9} \\ \binom{E}{\infty} & \binom{E}{23} & \binom{B}{15} & \binom{C}{19} & \binom{C}{13} \\ \binom{E}{\infty} & \binom{E}{10} & \binom{D}{21} & \binom{E}{\infty} & \binom{D}{19} \end{pmatrix}$$

Notice that in every case, the vertices chosen are exactly the same and, in the k-step matrix, the path lengths are exactly $\sum_{i=1}^{k} i = \frac{i(i+1)}{2}$ less than before. Why?

14.2 Setup

Assume that you are given the following problem: You are on one end of a line and there are 3 bit emitters numbered 1 to 3 that are emitting bits across the line. You are unable to tell at any given time which emitter is doing the sending. If p_i is the probability that emitter i emits a 0 (assuming it has access to the line), then $p_1 = .7, p_2 = .5, p_3 = .1$. Assume that the transition matrix for this system is

$$\begin{pmatrix} 0.5 & 0 & 0.5 \\ 0.2 & 0.6 & 0.2 \\ 1 & 0 & 0 \end{pmatrix}$$

Assume further that we start the system with emitter 2, and we measure the following sequence of bits at the other end of the line: 01101. What is the most likely sequence of emitters that emitted that bit sequence? What is the probability corresponding to that most likely sequence?

First, consider the most likely sequence that 01101 arose from. We are given that the sequence started in state 2. So we have 2-?-?-?-?. Now if the sequence started in state 2, it is mostly likely to remain there for one step, but there is only a 36% probability of staying in state 2 for 2 steps. So let's assume that we stay in state 2 for one step and then move out: 2-2-?-?-?. What is the most likely state to move to? The third bit is a 1 and emitter 3 likes to emit 1s so let's assume we move to state 3: 2-2-3-?-?. Once we are in state 3, we are required to head to state 1 (and we are backed up in that by noticing that the 4th bit is a 0, which emitter 1 likes to emit): 2-2-3-1-?. Finally, the last bit is a 1, making it more likely that we wind up in state 3 than state 1: 2-2-3-1-3. But this qualitative analysis is far from convincing...

15 Viterbi II

15.1 Theory

Claim: Given the Markov and sensor model assuptions,

$$P(s_0, \ldots, s_t, m_1, \ldots, m_t) = P(s_0) \prod_{i=1}^{t} P(s_i|s_{i-1}) P(m_i|s_i)$$

Proof:

$$P(s_0, \ldots, s_t, m_1, \ldots, m_t) = P(s_0, \ldots, s_t) P(m_1, \ldots, m_t|s_{0:t})$$

Concentrating on $P(s_0, \ldots, s_t)$:

$$P(s_0, \ldots, s_t) = P(s_0) P(s_1, \ldots, s_t|s_0) = P(s_0) P(s_1|s_0) P(s_2, \ldots, s_t|s_0, s_1) =$$

by the Markov assumption

$$P(s_0) \prod_{i=1}^{t} P(s_i|s_{i-1})$$

Concentrating on $P(m_1, \ldots, m_t|s_{0:t})$:

$$P(m_1, \ldots, m_t|s_{0:t}) = P(m_1|s_{0:t}) P(m_2, \ldots, m_t|s_{0:t}, m_1) =$$

$$P(m_1|s_1) P(m_2|s_{0:t}, m_1) P(m_3, \ldots, m_t|s_{0:t}, m_1, m_2) =$$

$$\prod_{i=1}^{t} P(m_i|s_i)$$

□

The goal is to minimize the information in the model and hence we look to minimize

$$-\log P(s_0, \ldots, s_t, m_1, \ldots, m_t) = -\log P(s_0) P(m_0|s_0) - \sum_{i=1}^{n} \log P(s_i|s_{i-1}) P(m_i|s_i)$$

We form the transition digraph after each step of the process; the shortest path in the graph corresponds to the minimum information.

15.2 Solving the example

Getting back to the example above: if there are 3 unknown states, then we assume that we are given a graph with 3 vertices. Vertex v_i will represent emitter i (for $i = 1, 2, 3$). There are two possibilities for a transition: either we arrive in a state where a 0 bit is being transmitted or a 1 bit is being

transmitted. Assume that we measure m (where m is either 0 or 1) after the completed transition; then the edge from v_a to v_b has weight equal to

$$-\log_2 P(s_{current} = b | s_{previous} = a) P(m_{current} = m | s_{current} = b)$$

which corresponds to the (a, b) entry of the matrix (assuming we are measuring information in bits).

The $m = 0$ transition matrix is

$$M^{(0)} = \begin{pmatrix} -\log_2(.5 * .7) & -\log_2(0 * .5) & -\log_2(.5 * .1) \\ -\log_2(.2 * .7) & -\log_2(.6 * .5) & -\log_2(.2 * .1) \\ -\log_2(1 * .7) & -\log_2(0 * .5) & -\log_2(0 * .1) \end{pmatrix} = \begin{pmatrix} 1.51457 & \infty & 4.32193 \\ 2.8365 & 1.73697 & 5.64386 \\ 0.514573 & \infty & \infty \end{pmatrix}$$

The $m = 1$ transition matrix is

$$M^{(1)} = \begin{pmatrix} -\log_2(.5 * .3) & -\log_2(0 * .5) & -\log_2(.5 * .9) \\ -\log_2(.2 * .3) & -\log_2(.6 * .5) & -\log_2(.2 * .9) \\ -\log_2(1 * .3) & -\log_2(0 * .5) & -\log_2(0 * .9) \end{pmatrix} = \begin{pmatrix} 2.73697 & \infty & 1.152 \\ 4.05889 & 1.73697 & 2.47393 \\ 1.73697 & \infty & \infty \end{pmatrix}$$

We also need to include a *penalty* matrix for starting in a given state. In this case, the (a, b) entry of the matrix will involve the "extra" penalty if the initial bit is m (where $m = 0$ or $m = 1$)

$$-\log_2 P(s_{initial} = a) P(m_{initial} = m | s_{initial} = a)$$

Notice that the column b does not appear; no transition is being made. So, in our case, we get

$$\begin{pmatrix} -\log_2(0 * .5) & -\log_2(0 * .5) & -\log_2(0 * .5) \\ -\log_2(1 * .5) & -\log_2(1 * .5) & -\log_2(1 * .5) \\ -\log_2(0 * .5) & -\log_2(0 * .5) & -\log_2(0 * .5) \end{pmatrix} = \begin{pmatrix} \infty & \infty & \infty \\ 1 & 1 & 1 \\ \infty & \infty & \infty \end{pmatrix}$$

We perform the following operations where $M_1 \circ M_2$ represents the min-plus multiplication of M_1 with M_2.

$$M^{(1)} \circ M^{(1)} = \begin{pmatrix} \binom{3}{2.88897} & \binom{3}{\infty} & \binom{1}{3.88897} \\ \binom{3}{4.2109} & \binom{2}{3.47393} & \binom{2}{4.2109} \\ \binom{1}{4.47393} & \binom{3}{\infty} & \binom{1}{2.88897} \end{pmatrix}$$

Then

$$(M^{(1)} \circ M^{(1)}) \circ M^{(0)} = \begin{pmatrix} \binom{3}{4.40354} & \binom{3}{\infty} & \binom{1}{7.2109} \\ \binom{3}{4.72547} & \binom{2}{5.2109} & \binom{1}{8.53282} \\ \binom{3}{3.40354} & \binom{3}{\infty} & \binom{1}{8.79586} \end{pmatrix}$$

One more transition...

$$((M^{(1)} \circ M^{(1)}) \circ M^{(0)}) \circ M^{(1)} = \begin{pmatrix} \binom{1}{7.14051} & \binom{3}{\infty} & \binom{1}{5.55554} \\ \binom{1}{7.46244} & \binom{2}{6.94786} & \binom{1}{5.87747} \\ \binom{1}{6.14051} & \binom{3}{\infty} & \binom{1}{4.55554} \end{pmatrix}$$

Finally, add in the penalty matrix to represent that the initial state is required to be state 2:

$$\begin{pmatrix} \infty & \infty & \infty \\ 8.46244 & 7.94786 & 6.87747 \\ \infty & \infty & \infty \end{pmatrix}$$

So the entry that represents the minimum information is clearly the $(2,3)$ entry, implying that we begin in state 2 and end in state 3: 2-?-?-?-3. Working backward from the 4-step matrix, we see that the final step came from 1: 2-?-?-1-3. Checking the 3-step matrix, we see that the shortest path from state 2 to state 1 has the final step at 3: 2-?-3-1-3. Finally, the 2-step matrix shows that the path from state 2 to state 3 goes through 2: 2-2-3-1-3.

Now what is the probability of this sequence? Notice that the information we receive from the model is 6.87747 bits so:

$$-\log p = 6.87748 \Rightarrow p \approx .0085$$

This is not a very high probability. Why is it so small? This is the probability of going through this exact state sequence AND of taking those measurements. Obviously, there are lots of possibilities so the probability of the most likely one decreases quickly.

16 Learning simple models

The basic idea behind learning is to build a model for an unknown function f. It will almost never be possible to determine f exactly, but if we assume that the model that we construct is correct, then all that is necessary is to learn the parameters that minimize the information that comes out of the model. Here we learn what happens if we assume that the model function f is extremely simple.

16.1 The urn/The binomial

Assume that you have an urn that contains red and blue balls. The goal is to find the fraction θ of red balls in the urn. You choose n balls from the urn (replacing the balls each time) and r of them come out red. What is your best guess for θ? Obviously, most people will answer $\theta \sim \frac{r}{n}$. Can you prove it?

What is the probability that you got r red balls and $n-r$ blue balls, given a value for θ? It is $\binom{n}{r}\theta^r(1-\theta)^{n-r}$. The information in our model is equal to the expression $-\log\binom{n}{r}\theta^r(1-\theta)^{n-r}$. Simplifying, taking the partial derivative with respect to θ and solving, we get exactly the fraction we think we should.

16.2 Why do we define mean and standard deviation of data the way we do?

The normal distribution is given by

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x-\mu}{2\sigma^2}}$$

The parameters of the model are σ and μ. Assume that you are given n points that were taken from this distribution $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$. We want to find the best values for the parameters given the independent measurements.

In order to do so, ask the following question: "How much information is coming out of our model?" The answer is the sum of the amount of information that comes out of each measurement (because we assumed that the measurements are independent). This expression is $\sum_{i=1}^{n} -\log P(x^{(i)})$. So the goal is to minimize the following function:

$$\sum_{i=1}^{n} -\log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^{(i)}-\mu}{2\sigma^2}} = n(\log\sqrt{2\pi} + \log\sigma) + \sum_{i=1}^{n} \frac{(x^{(i)}-\mu)^2}{2\sigma^2}$$

where I have chosen the base of the logarithm for convenience.

We can now take the partial derivatives of the above expressions with respect to both μ and σ to get the following results:

$$\mu = \frac{\sum_{i=1}^{n} x^{(i)}}{n}$$

and

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x^{(i)}-\mu)^2}{n}}$$

Not coincidentally, these are exactly the formulas that you will find in any standard statistics textbook for the best possible estimations for the average and standard deviation of a set of points.

16.3 Why do we use least squares?

Assume that you are taking measurements of some physical process and you know that for each data point x, you should get $f_\theta(x) + r$ where f is some function of the input x and unknown parameters θ (a vector) and r is random noise that is assumed to be Gaussian with mean 0 and variance σ. In practice, you don't usually assume there is only one source of error but many. However, because of the Central Limit Theorem, a theorem that states that the sum of a large number of independent identically distributed random variables approaches a Gaussian, this assumption is usually not so horrible. Let's assume that the experimenter knows what he "wants" the values of $f_\theta(x^{(i)})$ to be; in other words, let there be $y^{(i)}$ associated with each $x^{(i)}$ such that $y^{(i)} = f_\theta(x^{(i)}) + r^{(i)}$ so that the $y^{(i)}$ are to be thought of as the results of the experiment.

The normal distribution is given by

$$P_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The parameters of the model are σ and μ (and $\mu = 0$ in our case). Assume that you are given n data points that were drawn from this distribution $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$. We want to find the best values for the parameters θ given the independent measurements. In order to do so, ask the following question: "How much information is coming out of our model?" The answer is the sum of the amount of information that comes out of each measurement (because we assumed that the measurements are independent). Let us start with the standard probability

$$-\log P(x^{(1)}, \ldots, x^{(n)}, y^{(1)}, \ldots, y^{(n)}) =$$

$$-\log P(y^{(1)}, \ldots, y^{(n)} | x^{(1)}, \ldots, x^{(n)}) P(x^{(1)}, \ldots, x^{(n)}) =$$

$$-\log P(y^{(1)}, \ldots, y^{(n)} | x^{(1)}, \ldots, x^{(n)}) - \log P(x^{(1)}, \ldots, x^{(n)})$$

Notice now that the second term is completely independent of the model. It simply describes the distribution over which we are picking our samples. Thus, we can think of this term as a constant with respect to the model parameters and ignore it. This leaves us with the term (because the measurements are assumed to be made independently)

$$-\log P(y^{(1)}, \ldots, y^{(n)} | x^{(1)}, \ldots, x^{(n)}) = -\sum_{i=1}^{n} \log P(y^{(i)} | x^{(i)})$$

By the definitions given above, we know

$$y^{(i)} = f_\theta(x^{(i)}) + r^{(i)} \Rightarrow y^{(i)} - f_\theta(x^{(i)}) = r^{(i)}$$

So the quantity $y^{(i)} - f_\theta(x^{(i)})$ should be Gaussian with mean 0.

So the goal is to minimize the following function:

$$\sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y^{(i)} - f_\theta(x^{(i)}))^2}{2\sigma^2}} = n(\log\sqrt{2\pi} + \log\sigma) + \sum_{i=1}^{n} \frac{(y^{(i)} - f_\theta(x^{(i)}))^2}{2\sigma^2}$$

where I have chosen the base of the logarithm to be e for convenience. Note that σ and n are constants and this is therefore equivalent to minimizing the function

$$\sum_{i=1}^{n} (y^{(i)} - f_\theta(x^{(i)}))^2$$

The upshot of this analysis is that the standard sum of squares minimization is *exactly* the quantity we should be trying to minimize in most reasonable situations.

16.3.1 Locally weighted regression

When creating a prediction for some new value x using least squares, we first create the expression

$$\sum_{i=1}^{n}(y^{(i)} - f_\theta(x^{(i)}))^2$$

minimize this expression with respect to θ, and then output $f_\theta(x)$. Instead, we can introduce weights $w^{(i)}$ that depend inversely on the distance that $x^{(i)}$ is from x. We then can minimize

$$\sum_{i=1}^{n} w^{(i)}(y^{(i)} - f_\theta(x^{(i)}))^2$$

This is called *locally weighted regression*. The good feature of this is that you treat points closer to x with more importance than points far away from x. The bad feature of this is that you need to recalculate your model for each value of x.

17 Unsupervised learning: Expectation minimization

17.1 Setup

Oftentimes in practice, you are required to make classifications based on very imperfect information. Take the following, for example: Assume that you believe that you have points in the plane that are being generated from Gaussian sources. Assume that we use k different Gaussians for our model (where k is a known fixed constant) and that the training sample contains m different sample points (where m is a known fixed constant) $\{x^{(1)}, \ldots, x^{(m)}\}$. Our model chooses the jth Gaussian with *unknown* probability θ_j where $j \in \{1, 2, \ldots, k\}$ (such that $\sum_{j=1}^{k} \theta_j = 1$) and then randomly chooses a point in the plane with associated distribution equal to that of the jth Gaussian. Let $y^{(i)}$ be the label (an integer between 1 and k) of the *unknown* Gaussian distribution that the ith particle was chosen from. Our goal is to determine the model (i.e. find the best values for $\mu_i, \Sigma_i, \theta_i$) that best corresponds to the data; note that we are doing this *without* the values of any of the $y^{(i)}$! The $y^{(i)}$ are therefore *hidden*; if we knew what they were, life would be much simpler...

The setup for an EM-algorithm is as follows. You are given a data set $x^{(i)}$ for a given process for which there are hidden variables $y^{(i)}$ that are not measurable. The mathematical model is given with parameter vector θ.

17.1.1 Why direct action fails

Let us first see why solving this model from scratch is not feasible.In this case, our values $y^{(i)}$ are not given to us in the training data; we are attempting to

determine a distribution for them on the fly. So whereas normally we would get

$$\sum_{i=1}^{m} -\log P(x^{(i)}|y^{(i)})P(y^{(i)})$$

we instead wind up with

$$\sum_{i=1}^{m} -\log \sum_{j=1}^{k} P(x^{(i)}|y^{(i)}=j)P(y^{(i)}=j)$$

If you try to find a closed-form solution for this, you wil probably fail.

17.1.2 The algorithm

The algorithm is as follows:

1. Initialize any probability distribution for the $y^{(i)}$ and θ values.
2. Perform the following steps until convergence.
   (a) Expectation step: *Assuming the hidden parameters and distribution are correct*, calculate the expected information coming out of the model.
   (b) Minimization step: Minimize the information coming out of the model using the expression you found in the previous step.

17.2 Derivation

Lemma: Let p, q be probability distributions. Then $\sum_x p(x) \ln p(x) \geq \sum_x p(x) \ln q(x)$.

Proof: Note that $\forall x, \ln x \leq x - 1 \Rightarrow$

$$\sum_x p(x) \ln \frac{q(x)}{p(x)} \leq \sum_x p(x) \left(\frac{q(x)}{p(x)} - 1 \right) = \sum_x q(x) - \sum_x p(x) = 0$$

□

In what follows, x should be thought of as the data, y as hidden data, and θ as the model parameters.

Theorem: Let x be data, y be hidden variables, and θ be unknown parameters. Then

$$\sum_y P_{\theta'}(y|x) I_\theta(x,y) < \sum_y P_{\theta'}(y|x) I_{\theta'}(x,y) \Rightarrow I_\theta(x) < I_{\theta'}(x)$$

Proof: First note that $I_\theta(x,y) = -\ln P_\theta(x,y)$ and similarly for θ'. Assume that

$$\sum_y P_{\theta'}(y|x) I_\theta(x,y) < \sum_y P_{\theta'}(y|x) I_{\theta'}(x,y)$$

Consider the following expression.

$$\ln P_{\theta'}(x) - \ln P_{\theta}(x) = \sum_y P_{\theta'}(y|x) \ln P_{\theta'}(x) - \sum_y P_{\theta'}(y|x) \ln P_{\theta}(x)$$

$$= \sum_y P_{\theta'}(y|x) \ln \frac{P_{\theta'}(x)}{P_{\theta'}(y,x)} P_{\theta'}(y,x) - \sum_y P_{\theta'}(y|x) \ln \frac{P_{\theta}(x)}{P_{\theta}(y,x)} P_{\theta}(y,x)$$

$$= \sum_y P_{\theta'}(y|x) \ln \frac{P_{\theta'}(y,x)}{P_{\theta'}(y|x)} - \sum_y P_{\theta'}(y|x) \ln \frac{P_{\theta}(y,x)}{P_{\theta}(y|x)}$$

$$= \sum_y P_{\theta'}(y|x) \ln P_{\theta'}(y,x) - \sum_y P_{\theta'}(y|x) \ln P_{\theta}(y,x) + \left(\sum_y P_{\theta'}(y|x) P_{\theta}(y|x) - \sum_y P_{\theta'}(y|x) \ln P_{\theta'}(y|x) \right)$$

By the Lemma above, we have

$$\leq \sum_y P_{\theta'}(y|x) \ln P_{\theta'}(y,x) - \sum_y P_{\theta'}(y|x) \ln P_{\theta}(y,x)$$

$$= \sum_y P_{\theta'}(y|x) I_{\theta}(y,x) - \sum_y P_{\theta'}(y|x) I_{\theta'}(y,x) < 0$$

where the final conclusion is from the assumption. Thus we have that

$$\ln P_{\theta'}(x) - \ln P_{\theta}(x) < 0 \Rightarrow I_{\theta}(x) < I_{\theta'}(x)$$

The interpretation of this theorem should be as follows: As long as our new parameter value θ lowers the expected information assuming the old value of the parameters θ' are used to estimate the distribution of the hidden variables, we wind up with a better model.

17.3 EM Algorithm for Gaussian mixtures

Claim: $\nabla_x x^T A x = (A + A^T)x$ (where x is a column vector)

Proof: Let's track what multiplies x_i in the expression $x^T A x$. The jth entry of $x^T A$ is $\sum_{k=1}^n x_k A_{kj}$. Thus, $x^T A x = \sum_{j=1}^n \sum_{k=1}^n x_k A_{kj} x_j$. Now the ith coordinate of $\nabla_x x^T A x$ will be the following expression.

$$\frac{\partial \sum_{j=1}^n \sum_{k=1}^n x_k A_{kj} x_j}{\partial x_i}$$

In this double sum, for each term, either (a) $j = i, k \neq i$ or (b) $k = i, j \neq i$ or (c) $j = k = i$. Therefore, we get the following.

$$\nabla_x x^T A x = \sum_{k \neq i} x_k A_{ki} x_i + \sum_{j \neq i} x_i A_{ij} x_j + 2 A_{ii} x_i =$$

$$\sum_{k=1}^n A_{ki} x_k + \sum_{k=1}^n A_{ik} x_k = (A + A^T)x$$

□

Definition: The derivative $\frac{\partial f(A)}{\partial A}$ is defined to be the matrix with the same dimensions as A such that the i, j entry of the answer is equal to $\frac{\partial f(A)}{\partial A_{ij}}$.

Unproved claim: $\frac{\partial x^T A^{-1} x}{\partial A} = -A^{-T} x \cdot x^T A^{-T}$ (Note that $x \cdot x^T$ is different from $x^T x$. The latter has dimensions one by one and the former has dimensions $|x| \times |x|$.)

For example, take the given situation in d-dimensions (each Gaussian contains a mean/center of mass and a standard deviation/covariance matrix as hidden variables as well). Each Gaussian (μ_i, Σ_i) has the form

$$\mathcal{N}_{\mu_i, \Sigma_i}(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$$

where the μ_i are d-vectors and the Σ_i are $d \times d$-matrices. It will be a useful fact to note later that the covariance matrices Σ_i must all be symmetric.

Notice that the parameter vector actually consists of several different unknowns: μ_i (the means), Σ_i (the covariance matrices), θ_i (the distribution for which Gaussian gets chosen by the process). The $y^{(i)}$ are the hidden variables and the $x^{(i)}$ are the data values (constants).

Rather than calculating the exact values for the information, calculate the expected information based on the probability that your assumptions are correct given the model parameters.

1. For each i, j, we need to calculate the probability that $y^{(i)} = j$ assuming the value of all the other parameters are constant and correct; we want to find $P(y^{(i)} = j | x^{(i)}, \mu, \Sigma, \theta)$. To do this, we use Bayes Rule. Notice that

$$P(y^{(i)} = j | x^{(i)}, \mu, \Sigma, \theta) = \frac{P(x^{(i)} | y^{(i)} = j, \mu, \Sigma, \theta) P(y^{(i)} = j | \mu, \Sigma, \theta)}{P(x^{(i)} | \mu, \Sigma, \theta)}$$

   Unfortunately, $P(x^{(i)} | \mu, \Sigma, \theta)$ is problematic, we need to condition it on parameters that we know:

$$P(x^{(i)} | \mu, \Sigma, \theta) = \sum_{r=1}^{k} P(x^{(i)} | y^{(i)} = r, \mu, \Sigma, \theta) P(y^{(i)} = r | \mu, \Sigma, \theta)$$

   This gives the final expression

$$P(y^{(i)} = j | x^{(i)}, \mu\Sigma, \theta) = \frac{P(x^{(i)} | y^{(i)} = j, \mu, \Sigma, \theta) P(y^{(i)} = j | \mu, \Sigma, \theta)}{\sum_{r=1}^{k} P(x^{(i)} | y^{(i)} = r, \mu, \Sigma, \theta) P(y^{(i)} = r | \mu, \Sigma, \theta)}$$

   Now each of these terms involves only known parameters.

$$P(y^{(i)} = j | x^{(i)}, \mu, \Sigma, \theta) = \frac{\frac{1}{\sqrt{(2\pi)^d |\Sigma_j|}} e^{-\frac{1}{2}(x^{(i)}-\mu_j)^T \Sigma_j^{-1}(x^{(i)}-\mu_j)} \theta_j}{\sum_{r=1}^{k} \frac{1}{\sqrt{(2\pi)^d |\Sigma_r|}} e^{-\frac{1}{2}(x^{(i)}-\mu_r)^T \Sigma_r^{-1}(x^{(i)}-\mu_r)} \theta_r}$$

Notice that every parameter in the above formula is assumed to be correct and constant. We can plug these in and determine the current values of $P(y^{(i)} = j|x^{(i)}, \mu, \Sigma, \theta)$ via the above formula. Make the following definition:

$$P(y^{(i)} = j|x^{(i)}, \mu, \Sigma, \theta) \equiv \gamma_{ij}$$

where the γ_{ij} are constants. Now we calculate the expected information coming out of the model given the model parameters as follows.

$$\sum_{i=1}^{m}\sum_{j=1}^{k} P(y^{(i)} = j|x^{(i)}, \mu, \Sigma, \theta) I(x, \mu, \Sigma, \theta|y^{(i)} = j, \mu, \Sigma, \theta) =$$

$$\sum_{i=1}^{m}\sum_{j=1}^{k} \gamma_{ij} I(x^{(i)}, y^{(i)} = j) =$$

$$\sum_{i=1}^{m}\sum_{j=1}^{k} \gamma_{ij}(-\log P(x^{(i)}, y^{(i)} = j)) =$$

$$\sum_{i=1}^{m}\sum_{j=1}^{k} \gamma_{ij}(-\log P(x^{(i)}|y^{(i)} = j)P(y^{(i)} = j)) =$$

$$\sum_{i=1}^{m}\sum_{j=1}^{k} \gamma_{ij}\left(-\log \frac{1}{\sqrt{(2\pi)^d|\Sigma_j|}} e^{-\frac{1}{2}(x^{(i)}-\mu_j)^T\Sigma_j^{-1}(x^{(i)}-\mu_j)}\theta_j\right)$$

2. • Given the expression above, we can now maximize with respect to μ, Σ, θ. Let's minimize with respect to μ_a.

$$\frac{\partial}{\partial \mu_a}\sum_{i=1}^{m}\sum_{j=1}^{k} \gamma_{ij}\left(-\log \frac{1}{\sqrt{(2\pi)^d|\Sigma_j|}} e^{-\frac{1}{2}(x^{(i)}-\mu_j)^T\Sigma_j^{-1}(x^{(i)}-\mu_j)}\theta_j\right) =$$

$$\frac{\partial}{\partial \mu_a}\sum_{i=1}^{m}\sum_{j=1}^{k} \gamma_{ij}\frac{1}{2}(x^{(i)}-\mu_j)^T\Sigma_j^{-1}(x^{(i)}-\mu_j) =$$

$$\sum_{i=1}^{m}\frac{1}{2}\gamma_{ia}\frac{\partial}{\partial \mu_a} 2\mu_a^T\Sigma_a^{-1}x^{(i)} - \mu_a^T\Sigma_a^{-1}\mu_a =$$

Using claim 1,

$$\sum_{i=1}^{m}\gamma_{ia}\left(\Sigma_a^{-1}x^{(i)} - \frac{1}{2}(\Sigma_a^{-1} + (\Sigma_a^{-1})^T)\mu_a\right) =$$

Recalling the symmetry of the covariance matrix

$$\sum_{i=1}^{m}\gamma_{ia}\left(\Sigma_a^{-1}x^{(i)} - \Sigma_a^{-1}\mu_a\right)$$

If we set this expression equal to 0, we get that

$$\mu_a = \frac{\sum_{i=1}^{m} \gamma_{ia} x^{(i)}}{\sum_{i=1}^{m} \gamma_{ia}}$$

- We can also take the derivative with respect to θ_a. Recall that we are constrained by the fact that θ represents a distribution vector and thus its components must sum to 1. We must there utilize a Lagrangian.

$$\frac{\partial}{\partial \theta_a} \sum_{i=1}^{m} \sum_{j=1}^{k} \gamma_{ij} \left(-\log \frac{1}{\sqrt{(2\pi)^d |\Sigma_j|}} e^{-\frac{1}{2}(x^{(i)}-\mu_j)^T \Sigma_j^{-1} (x^{(i)}-\mu_j)} \theta_j \right) + \alpha(\sum_{i=1}^{k} \theta_i - 1) =$$

$$\sum_{i=1}^{m} (\frac{\gamma_{ia}}{\theta_a} + \alpha) = 0 \Rightarrow \forall a, \theta_a = -\frac{\sum_{i=1}^{m} \gamma_{ia}}{\alpha}$$

Now note that

$$\sum_{j=1}^{k} \theta_j = 1 \Rightarrow \sum_{j=1}^{k} -\frac{\sum_{i=1}^{m} \gamma_{ij}}{\alpha} = 1 \Rightarrow -\frac{\sum_{i=1}^{m} 1}{\alpha} = 1 \Rightarrow \alpha = -m \Rightarrow$$

$$\theta_a = \frac{\sum_{i=1}^{m} \gamma_{ia}}{m}$$

- Finally, we take the derivative of the information function with respect to Σ_a.

$$\frac{\partial}{\partial \Sigma_a} \sum_{i=1}^{m} \sum_{j=1}^{k} \gamma_{ij} \left(-\log \frac{1}{\sqrt{(2\pi)^d |\Sigma_j|}} e^{-\frac{1}{2}(x^{(i)}-\mu_j)^T \Sigma_j^{-1} (x^{(i)}-\mu_j)} \theta_j \right) =$$

By claim 2,

$$\sum_{i=1}^{m} \frac{1}{2} \gamma_{ia} \left(\Sigma_a^{-1} - \frac{1}{2} \Sigma_a^{-1} (x^{(i)} - \mu_a)(x^{(i)} - \mu_a)^T \Sigma_a^{-1} \right) = 0 \Rightarrow$$

$$\Sigma_a = \frac{\sum_{i=1}^{m} \gamma_{ia} (x^{(i)} - \mu_a)(x^{(i)} - \mu_a)^T}{\sum_{i=1}^{m} \gamma_{ia}}$$

17.4 Example

To illustrate this process, 2000 points with $x-$ and y-coordinates between the values of -3 and 9. Three skewed normal distributions were generated with the following parameters.

$$\mu_1 = (1, 1), \Sigma_1 = \begin{pmatrix} 1 & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} & 1 \end{pmatrix}$$

$$\mu_2 = (2, 7), \Sigma_2 = \begin{pmatrix} 4 & \sqrt{2} \\ \sqrt{2} & 1 \end{pmatrix}$$
$$\mu_3 = (5, 3), \Sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The numbers were generated with the first distribution with probability $\frac{1}{5}$, second distribution probability $\frac{3}{10}$, and third distribution with probability $\frac{1}{2}$. The EM algorithm was initialized with all μ_i taking uniformly random choices for the x and y coordinates between -3 and 9 and all Σ the identity matrix. Finally, the θ was initialized to $\frac{1}{3}$ for each distribution.

18 Learning theory

Mathematically, there is some unknown function f that we are attempting to model, but that we have information about other than the value of the function on the given points (i.e. that for $i = 1, 2, \ldots, m$, $f(x^{(i)}) = y^{(i)}$). We will attempt to approximate f via some hypothesis function h that we will choose from a space of hypotheses $\mathcal{H}$. Note that there are *two* choices being made here, *not one*; we need to choose both $\mathcal{H}$ and then $h \in \mathcal{H}$. It is important to note that this slightly differs from the situations we will encounter in that we will have foreknowledge of certain aspects of the function: we never will have to choose $\mathcal{H}$ because it will be chosen for us.

We are going to perform an analysis that will clarify why making the choice of $\mathcal{H}$ judiciously is important.

- Note that if we keep $\mathcal{H}$ small, then it will be easy to find a hypothesis $h \in \mathcal{H}$ that comes closest to f, given the data. However, if $\mathcal{H}$ is small enough, then we may not be able to fit the data we are given very well...
- You might wonder why we don't choose $\mathcal{H}$ to be the space of all possible functions so as to keep f inside the hypothesis space: Intuitively, if we allow ourselves latitude to choose any function, then, given only the data that we have, there are *multiple* possible choices for h that are likely to match f exactly and there will be no way to differentiate between them. We may wind up with a function that matches the data perfectly and yet does not generalize to data points outside the sample.

Note that when we perform our learning algorithm, we will be choosing our hypothesis from $\mathcal{H}$ based on the data set $\mathcal{D}$. Let us define the hypothesis that arises from the data set $\mathcal{D}$ after our learning algorithm to be $h_{\mathcal{D}}$. Note that there is an explicit dependence on the data set that is chosen from the space of all possible data sets. Unfortunately, in any normal application, we are not allowed to choose the data set that we want; it is simply drawn from an outside distribution over which we have no knowledge or control. We *do* have control over $\mathcal{H}$...

Let us define the expected out-of-sample squared error, given a hypothesis space $\mathcal{H}$ (that we have chosen), to be the expected squared error that we get

from any point in the space. It is our overall goal to minimize this quantity. Notice that we need to average not only all points in the space but also over all possible data sets $\mathcal{D}$:

$$E_x E_{\mathcal{D}}(h_{\mathcal{D}}(x) - f(x))^2$$

There is an interesting entity hidden within this expression. Define the *average hypothesis* to be the hypothesis that arises when averaging over all possible data sets:

$$E_{\mathcal{D}} h_{\mathcal{D}} \equiv \overline{h}$$

so that for any $x, \overline{h}(x) = E_{\mathcal{D}} h_{\mathcal{D}}(x)$.

Now, we have the following sequence of equalities.

$$E_x E_{\mathcal{D}}(h_{\mathcal{D}}(x) - f(x))^2 =$$

$$E_x E_{\mathcal{D}}\left[(h_{\mathcal{D}}(x) - \overline{h}(x)) + (\overline{h}(x) - f(x))\right]^2 =$$

$$E_x E_{\mathcal{D}}\left[(h_{\mathcal{D}}(x) - \overline{h}(x))^2 + (\overline{h}(x) - f(x))^2 + 2(h_{\mathcal{D}}(x) - \overline{h}(x))(\overline{h}(x) - f(x))\right] =$$

$$E_x E_{\mathcal{D}}(h_{\mathcal{D}}(x) - \overline{h}(x))^2 + E_x E_{\mathcal{D}}(\overline{h}(x) - f(x))^2 +$$

$$E_x E_{\mathcal{D}} 2(h_{\mathcal{D}}(x) - \overline{h}(x))(\overline{h}(x) - f(x))$$

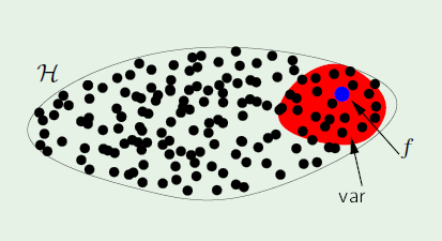
Concentrating on the final expression, notice that $E_{\mathcal{D}} h_{\mathcal{D}}(x) = \overline{h}(x)$ by definition. So the final term is equal to 0. The entire expression reduces to

$$E_x E_{\mathcal{D}}(h_{\mathcal{D}}(x) - \overline{h}(x))^2 + E_x E_{\mathcal{D}}(\overline{h}(x) - f(x))^2 =$$

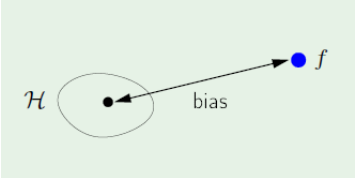
$$E_x E_{\mathcal{D}}(h_{\mathcal{D}}(x) - \overline{h}(x))^2 + E_x(\overline{h}(x) - f(x))^2$$

Interpreting this is nontrivial:

- The first term $E_x E_{\mathcal{D}}(h_{\mathcal{D}}(x) - \overline{h}(x))^2$ represents the expected squared difference between the hypothesis we choose and the average hypothesis. In some sense, this quantity represents the variability among the possible hypotheses that we have allowed into our set $\mathcal{H}$. Notice that in general, as $|\mathcal{H}|$ gets larger, this quantity will increase and vice versa.



- The second term $E_x(\overline{h}(x) - f(x))^2$ represents the expected squared difference between the average hypothesis that we expect to get with our choice of $\mathcal{H}$ and the true function f. In general, as $|\mathcal{H}|$ gets larger, this quantity will decrease and vice versa. This is called the bias because your choice of mathematical model (i.e. your own predetermined notion of what model the data *should* fit) has biased the outcome hypothesis.



So there is a clear tradeoff between wanting the hypothesis set to be large enough so that we don't introduce our own bias into the results yet small enough so that we have some hope of being able to zero in on a hypothesis that has a good chance of generalizing to data out of sample.

Throughout this class, we will be choosing the mathematical models we use based on foreknowledge of the process that is producing the output. However, in practical situations, this may not be feasible. For example, given a product, how would you accurately determine the best web page on which to buy advertising? Given a person applying for a loan, how would you accurately determine whether they should get a loan, and, if so, what should the terms of the loan be?

Interesting to think about: Humanity has been engaged in a multiple millenia-long experiment in machine learning where *we are the machines*. Since the time of Aristotle (and probably even before), we have attempted to predict the laws of nature not just to better and better accuracy but also with better models. Every time mathematicians come up with a potentially interesting new model, physicists enter it into humanity's ever-enlarging set $\mathcal{H}$ and start collecting data to test it. At the time of Aristotle, the set $\mathcal{H}$ was very small so it was fairly easy to fit the existing data; the bias for predictions was very large. Basically, whatever Aristotle assumed logically followed from a few very basic observations was considered the best model for the data. After Descartes, humanity was able to add many more accurate models to its prediction library and, after many observations, Newton came up with the best one to fit most of the data collected. However, up until the time of Einstein, Bohr, Heisenberg, etc., there was *too much variance* in our hypothesis set: we had a mathematical model that explained *most* of the data, but not *all* (i.e. Newtonian physics was breaking down when we measured objects moving very fast or objects were very small.). Our hypothesis set is still growing as mathematicians discover new ways to include mathematics into physical models, and whether we are at a point where we have too much bias or too much variance is still a topic of intense research: The existence of the Higgs boson was just "observed" in 2013, confirming, in many physicists minds, the standard model of particle physics. However, the standard model, though internally mathematically consistent, does not explain the full theory of gravitation as described by Einstein. So does humanity have all the mathematical models we need to describe what is "really" happening but just have not stumbled across the right one (i.e. Do we have a problem of too much variance?) or do mathematicians need to keep searching for more descriptive models for physicists to test (i.e. Do we have a problem of too much bias due to our relative mathematical ignorance as compared with humanity, say, 200 years in the future?)?

In order to keep $|\mathcal{H}|$ down without introducing too much bias, researchers have introduced the concept of the VC-dimension of the model space, but this subject technically belongs in a machine learning course and is therefore beyond the scope of this class.

19 Introduction to supervised learning: Logistic

19.1 Setup

In supervised learning, the setting will always be as follows: You will be given a sequence of m measurements that we will denote by $x^{(1)}, \ldots, x^{(m)}$. Usually, each of these $x^{(i)}$ values will be a column vector $(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})^T$. We will map each of these $x^{(i)}$ values to an "answer" $y^{(i)}$ which will very often be an element of $\{0, 1\}$. Based on these training samples, the goal is to determine, given another example x, determine the "correct" y value for it.

So what is the information leaving the model if $(\{x^{(1)}, y^{(1)}\}, \ldots, \{x^{(m)}, y^{(m)}\})$ are the data points?

$$-\log P(x^{(1)}, \ldots, x^{(n)}, y^{(1)}, \ldots, y^{(n)}) =$$
$$-\log P(y^{(1)}, \ldots, y^{(n)} | x^{(1)}, \ldots, x^{(n)}) P(x^{(1)}, \ldots, x^{(n)}) =$$
$$-\log P(y^{(1)}, \ldots, y^{(n)} | x^{(1)}, \ldots, x^{(n)}) - \log P(x^{(1)}, \ldots, x^{(n)})$$

Notice now that the second term is completely independent of the model. It simply describes the distribution over which we are picking our samples. Thus, we can think of this term as a constant with respect to the model parameters and ignore it. This leaves us with the term (because the measurements are assumed to be made independently)

$$-\log P(y^{(1)}, \ldots, y^{(n)} | x^{(1)}, \ldots, x^{(n)}) = -\sum_{i=1}^{n} \log P(y^{(i)} | x^{(i)})$$

19.2 Logistic

Throughout this section, we will measure the information function in base e because it will make calculations easier.

Assume that you are an extremely high-priced political consultant and one of your jobs is to create a model of whether a given person will or will not vote in the coming election. You can take as input data only publicly available information such as age, gender, annual income, etc. Assume that the input data vector is x. We will assume that the final entry in every vector is 1; this will allow us the use of a constant term in our linear model.

Due to its shape, we will assume that the model is as follows (where θ is the parameter vector of the model). Note that h stands for *hypothesis*.

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

Just to get an idea for how this function behaves, answer the following questions. As you are answering these questions, think about $\theta^T x$ as a linear transformation of the data that weights each input appropriately.

1. As $\theta x \to \infty$, what does $h_\theta(x)$ approach?
2. As $\theta x \to -\infty$, what does $h_\theta(x)$ approach?
3. If $\theta x = 0$, what is $h_\theta(x)$?
4. What are the high and low values of $h_\theta(x)$?

Based on this, it is reasonable to think of $h_\theta(x)$ as the probability that a given person with characteristics given by x will vote; in other words, $h_\theta(x) = P(y = 1|x)$.

Assume that we collect data from previous elections:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

where $y^{(i)} \in \{0, 1\}$. Then how can we find the best θ parameter to put into our model?

The information coming out of our model is $I(x, y, \theta) = \sum_{i=1}^{m} -\log P(y^{(i)}|x^{(i)})$. Note that $P(y^{(i)} = 1|x^{(i)}) = h_\theta(x^{(i)})$ and $P(y^{(i)} = 0|x^{(i)}) = 1 - h_\theta(x^{(i)})$. For mathematical convenience, we can write

$$P(y^{(i)}|x^{(i)}) = h_\theta(x^{(i)})^{y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \Rightarrow$$

$$I(x, y, \theta) = \sum_{i=1}^{m} -\log P(y^{(i)}|x^{(i)}) =$$

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

Assume that we want to minimize this information expression using the gradient technique with parameter α so that

$$\theta_{next} = \theta_{current} - \alpha \nabla_\theta I(x, y, \theta)$$

To evaluate this, we note that

$$\frac{\partial I(x, y, \theta)}{\partial \theta_j} = \sum_{i=1}^{m} -y^{(i)} \frac{\partial \log h_\theta(x^{(i)})}{\partial \theta_j} - (1 - y^{(i)}) \frac{\partial \log(1 - h_\theta(x^{(i)}))}{\partial \theta_j} =$$

$$\sum_{i=1}^{m} -y^{(i)} \frac{\partial \log h_\theta(x^{(i)})}{\partial h_\theta(x^{(i)})} \frac{\partial h_\theta(x^{(i)})}{\partial \theta x^{(i)}} \frac{\partial \theta x^{(i)}}{\partial \theta_j} - (1 - y^{(i)}) \frac{\partial \log(1 - h_\theta(x^{(i)}))}{\partial h_\theta(x^{(i)})} \frac{\partial h_\theta(x^{(i)})}{\partial \theta x^{(i)}} \frac{\partial \theta x^{(i)}}{\partial \theta_j}$$

Now we can easily verify the following equations.

$$\frac{\partial \log h_\theta(x^{(i)})}{\partial h_\theta(x^{(i)})} = \frac{1}{h_\theta(x^{(i)})}$$

$$\frac{\partial h_\theta(x^{(i)})}{\partial \theta x^{(i)}} = h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))$$

$$\frac{\partial \theta x^{(i)}}{\partial \theta_j} = x_j^{(i)}$$

Putting it all together...

$$\frac{\partial I(x, y, \theta)}{\partial \theta_j} =$$

$$\sum_{i=1}^{m} -y^{(i)} \frac{1}{h_\theta(x^{(i)})} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x_j^{(i)} - (1-y^{(i)}) - \frac{1}{1 - h_\theta(x^{(i)})} h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x_j^{(i)} =$$

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

$$-\sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)} \Rightarrow$$

$$\theta_{next} = \theta_{current} + \alpha \sum_{i=1}^{n} (y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$$

19.3 Example

There is an example of this process (logistic.gif,logistic.png). 50 data points each were generated using 2 two-dimensional Gaussian distributions, one with mean at (0,0) and the other with mean $(-\frac{1}{2}, \frac{1}{2})$. Note that the data actually overlaps and there will therefore be some error no matter what model you use. The pictures show the separation that the gradient descent method over a period of a few hundred iterations.

20 Supervised learning: Poisson

In June 1943 during the second World War the Germans launched a much feared secret weapon - a small, pilotless, jet-propelled plane carrying a ton of explosives that detonated on contact. The Germans called it the V1 Vergeltung (meaning reprisal) - the British called it the flying bomb.

Of the first 10 launched, 5 crashed almost immediately and only one found its way to Britain where it killed 6 people. However, subsequent flying bomb attacks on Britain caused widespread death and destruction. Within a year 1600 people had been killed by flying bombs. In the autumn of 1944 nearly 10000 V1 flying bombs were launched against British towns. The majority of those that got through descended on London. The German propaganda claimed the V1 was an accurately aimed weapon. The question for the British was: are these bombing patterns hits truly a result of spies in the Allied ranks or was there an alternative explanation?

During the Blitz, a chart was kept showing V-Bomb hits on South London. The chart was divided into 576 squares, each representing an area 0.25 km on a side. Of those 576 squares, the number with a given number of V-Bomb hits was: 1 square with 5 or more hits, 7 squares with 4 hits, 35 squares with 3 hits, 93 squares with 2 hits, and 211 squares with 1 hit, leaving 229 squares without any hits.

So the goal is to check just how accurate the flying bombs are. To do so, we will start with some interesting math facts.

Warmup: $\lim_{n\to\infty}(1+\frac{\alpha}{n})^n = e^\alpha$ □

Lemma 1:

$$\lim_{n\to\infty}\binom{n}{k}/n^k = \frac{1}{k!}$$

Proof:

$$\lim_{n\to\infty}\binom{n}{k}/n^k = \lim_{n\to\infty}\frac{n!}{k!(n-k)!n^k} = \lim_{n\to\infty}\frac{n(n-1)(n-2)\dots(n-k+1)}{n^k}\frac{1}{k!} = \frac{1}{k!}$$

□

Lemma 2:

$$\lim_{n\to\infty}\binom{n}{k}(\frac{x}{n})^k(1-\frac{x}{n})^{n-k} = e^{-x}\frac{x^k}{k!}$$

Proof: Recalling Lemma 1,

$$\lim_{n\to\infty}\binom{n}{k}(\frac{x}{n})^k(1-\frac{x}{n})^{n-k} = \lim_{n\to\infty}\frac{x^k}{k!}(1-\frac{x}{n})^{n-k} = \lim_{n\to\infty}\frac{x^k}{k!}(1-\frac{x}{n})^{-k}e^{-x} = e^{-x}\frac{x^k}{k!}$$

□

Now, let's try a though experiment. Assume that during a certain lengthy period of time, we know that the expected number of bomb hits in a given grid square is some constant number. In our case, that expected number is slightly more than 0.9 (maybe around .99323); call it μ. Now slice up that lengthy period of time into n pieces. The probability of a bomb hit during any of those time slices is our average number of bomb hits per square μ divided by n; we are ignoring the possibility of 2 or more bombs hitting a given grid square in a given time slice because the probability of doing so is so low and, more importantly, because we cannot distinguish between two explosions in a given second and one. Now what is the probability that we get k hits in a given grid square in the lengthy time period? That is the probability that k "success" trials occur out of n trials: $\binom{n}{k}(\frac{\mu}{n})^k(1-\frac{\mu}{n})^{n-k}$. As $n\to\infty$, by Lemma 2, this probability is $e^{-\mu}\frac{\mu^k}{k!}$.

Probabilities for $r = 0.9$	Expected number of hits	Actual number of hits
$P(k = 0) = 0.4066$	234	229
$P(k = 1) = 0.3659$	211	211
$P(k = 2) = 0.1647$	95	93
$P(k = 3) = 0.0494$	28	35
$P(k = 4) = 0.0111$	6	7
$P(k = 5, 6, \dots) = 0.0023$	1	1

Note that these values track almost perfectly with a random Poisson distribution, indicating that the Germans were perhaps being slightly less than truthful with their military advertising.

Now, we will return to the subject at hand. We are no longer dealing with a single city (London) in a single country (England). We are now dealing with the Allies versus the Axis powers and are analyzing the concentrating of Axis bombing forces being allocated to various cities under Allied control. Unfortunately, the data that you have collected does not separate neatly into cities; the Germans have somehow gotten to your data and corrupted that field. Instead, you only have accurate feature vectors $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ of the small plots of land on which the bombs are dropping and $y^{(1)}, y^{(2)}, \dots, y^{(m)}$ corresponding to the number of bombs that landed on each plot. We have already been convinced that the process for a given city is certainly a Poisson process, and we wish to build as accurate a model as we can of the bombing strategy of the enemy. Assuming that the parameter vector θ interacts linearly with the feature vector, we will assume that

$$\forall k \geq 0, P(y = k|x) = e^{-(\theta x)^2} \frac{(\theta x)^{2k}}{k!}$$

The reason for the square on the linear transformation is that the mean of the Poisson process is required to be positive. Thus, given a feature vector x and parameters θ, our hypothesis for the expected number of bombs dropped on a particular grid square is

$$h_\theta(x) = \sum_{k=0}^{\infty} e^{-(\theta x)^2} \frac{(\theta x)^{2k}}{k!} = (\theta x)^2$$

So we are looking to minimize the quantity (again, with the base of the logarithm chosen to be e for convenience)

$$I(x, y, \theta) = \sum_{i=1}^{n} -\log P(y^{(i)}|x^{(i)}) =$$

$$\sum_{i=1}^{n} -\log e^{-(\theta x^{(i)})^2} \frac{(\theta x^{(i)})^{2y^{(i)}}}{y^{(i)}!} =$$

$$\sum_{i=1}^{n} (\theta x^{(i)})^2 - 2y^{(i)} \log \theta x^{(i)} + \log(y^{(i)}!)$$

600 grid squares were generated in each of 3 cities: London, Stalingrad, and Hiroshima. There were two major differentiating features in the feature vector (along with the standard constant). The two features in the feature vector x were (a) the average propensity of the residents on the given grid square to wear large, silly hats (b) the total number of establishments within the given grid square selling crumpets. Thus, a given grid square in each city looks as follows:

- London: (a) Gaussian with mean 2.5, (b) Gaussian with mean 3, (result) Poisson with mean 1
- Stalingrad: (a) Gaussian with mean 2, (b) Gaussian with mean 2, (result) Poisson with mean 40
- Hiroshima: (a) Gaussian with mean 1, (b) Gaussian with mean 1, (result) Poisson with mean 100

A numerical analysis package was used to minimize the information function directly (poisson.png, poissonResult.png).

21 Generative models: Gaussians

In this section, we will explore a different approach to classification. Previously, if we wanted to learn to distinguish between two different categories, we would collect a feature vector x of the item. We then trained a model whose job it was to distinguish one category from the other and simply plugged the vector x into the model that spit out a response. These types of models are called *discriminative learning algorithms.*

Consider the following approach: Given a training set, build a toy model of the first category and the second category. Then, compare any new feature vector x with the toy models and check which of the two it most resembles. Output the result. These algorithms are called *generative learning algorithms.*

Recall that the point of the information minimization is to minimize the quantity $\sum_{i=1}^{m} -\log P(x^{(i)}, y^{(i)})$. Up until now in the course, we have use Bayes Rule to transform this quantity into $\sum_{i=1}^{m} -\log P(y^{(i)}|x^{(i)})P(x^{(i)}) = \sum_{i=1}^{m} -\log P(y^{(i)}|x^{(i)}) + \sum_{i=1}^{m} -\log P(x^{(i)})$. We then noticed that the second term is simply a constant because the $x^{(i)}$ are given in the problem and so we just minimized $\sum_{i=1}^{m} -\log P(y^{(i)}|x^{(i)})$. We will now turn this around:

$$\sum_{i=1}^{m} -\log P(x^{(i)}, y^{(i)}) = \sum_{i=1}^{m} -\log P(x^{(i)}|y^{(i)})P(y^{(i)})$$

Let's take, for example, two Gaussian distributions on the real line. You may assume that there are two twin shooters (who shoot with exactly the same accuracy but possibly different frequency) aiming for two different points on the line. Let the distributions for $i = 0, 1$ be given by

$$p^{(i)}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu_i)^2}{2\sigma^2}}$$

Assume that whenever a point it measured, it comes from Gaussian 0 with probability ϕ and Gaussian 1 with probability $1-\phi$. All of the quantities $\mu_0, \mu_1, \sigma, \phi$ are all parameters of the model.

Recall that the total information is equal to (logarithm is base e for convenience)

$$I(x, y, \mu_0, \mu_1, \sigma, \phi) = \sum_{i=1}^{m} -\log P(y^{(i)}, x^{(i)}) = \sum_{i=1}^{m} -\log P(x^{(i)}|y^{(i)})P(y^{(i)}) =$$

$$\sum_{i=1}^{m} -\log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x^{(i)}-(\mu_0(1-y^{(i)})+\mu_1 y^{(i)}))^2}{2\sigma^2}} \phi^{1-y^{(i)}}(1-\phi)^{y^{(i)}} =$$

$$\sum_{i=1}^{m} \frac{1}{2}\log 2\pi + \log\sigma + \frac{(x^{(i)}-(\mu_0(1-y^{(i)})+\mu_1 y^{(i)}))^2}{2\sigma^2} - (1-y^{(i)})\log\phi - y^{(i)}\log(1-\phi)$$

Taking the partial derivative with respect to μ_0, we get

$$\frac{\partial I}{\partial \mu_0} = \sum_{i=1}^{m} \frac{(y^{(i)}-1)(x^{(i)}+(y^{(i)}-1)\mu_0 - y^{(i)}\mu_1)}{\sigma^2} = 0 \Rightarrow$$

$$\sum_{i=1}^{m} (y^{(i)}-1)(x^{(i)}+(y^{(i)}-1)\mu_0 - y^{(i)}\mu_1) = 0 \Rightarrow$$

$$\sum_{i:y^{(i)}=0} (x^{(i)}+(y^{(i)}-1)\mu_0 - y^{(i)}\mu_1) = 0 \Rightarrow$$

$$\sum_{i:y^{(i)}=0} x^{(i)} = \mu_0 \sum_{i:y^{(i)}=0} 1 \Rightarrow \mu_0 = \frac{\sum_{i:y^{(i)}=0} x^{(i)}}{\sum_{i:y^{(i)}=0} 1}$$

Taking the partial derivative with respect to μ_1, we get

$$\frac{\partial I}{\partial \mu_1} = \sum_{i=1}^{m} -\frac{y^{(i)}(x^{(i)}+(y^{(i)}-1)\mu_0 - \mu_1 y^{(i)})}{\sigma^2} = 0 \Rightarrow$$

$$\sum_{i=1}^{m} y^{(i)}(x^{(i)}+(y^{(i)}-1)\mu_0 - \mu_1 y^{(i)}) = 0 \Rightarrow$$

$$\sum_{i:y^{(i)}=1} (x^{(i)}-\mu_1) = 0 \Rightarrow \mu_1 = \frac{\sum_{i:y^{(i)}=1} x^{(i)}}{\sum_{i:y^{(i)}=1} 1}$$

Taking the partial derivative with respect to ϕ, we get

$$\frac{\partial I}{\partial \phi} = \sum_{i=1}^{m} \frac{1-\phi-y^{(i)}}{\phi(\phi-1)} = 0 \Rightarrow$$

$$\sum_{i=1}^{m} 1 - \phi - y^{(i)} = 0 \Rightarrow \phi = \frac{\sum_{i:y^{(i)}=0} 1}{m}$$

Taking the partial derivative with respect to σ, we get

$$\frac{\partial I}{\partial \sigma} = \sum_{i=1}^{m} \frac{\sigma^2 - (x^{(i)} + (y^{(i)} - 1)\mu_0 - y^{(i)}\mu_1)^2}{\sigma^2} = 0 \Rightarrow$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{m}(x^{(i)} + (y^{(i)} - 1)\mu_0 - y^{(i)}\mu_1)^2}{m}}$$

Notice that each of these conclusions is exactly the conclusion you would have made on your own had you been forced to make a "common sense" judgement.

22 Naive Bayes

Assume that you are trying to create a spam classifier for email. The goal is, whenever a person is sent an email, for the AI to examine the text of the email and, based on that text, make a probabilistic decision as to whether the email is spam or not. You may assume that you have a corpus of prior emails that have already been correctly classified by the user(s) of the system. As per usual, we will assume that the input data (which has yet to be defined) is $(x^{(1)}, \ldots, x^{(m)})$ and the output data is $(y^{(1)}, \ldots, y^{(m)})$ where $\forall i, y^{(i)} \in \{0, 1\}$.

First, gather together all of the words that exist in the corpus and create a dictionary. Let the number of words in the dictionary be d. List them in alphabetical order. For the ith example in the data set, define $x^{(i)}$ to be a d-dimensional 0-1 vector. Let the jth coordinate of $x^{(i)}$ be 1 if the jth word of the ith email contains word j at least once and 0 otherwise. Note that this is a very large vector of mostly 0's for each email.

This will be a generative model and hence we will be interested in the two distributions $P(x|y)$ and $P(y)$. The second is relatively straightforward, but the first is problematic. What is the probability of a set of words occuring given that an email is or is not spam? We will make a very strong and wildly incorrect assumption that seems, for some reason, to work well in practice: Given two distinct words in the dictionary, we will assume that, given knowledge of whether the email is or is not spam, the probability that the two words appear in a given email is independent. Essentially what we are assuming is that there is a "good" email generator that chooses words independently from one distribution and a "bad" (spam) email generator that chooses words independently from a different distribution. Once we decide whether we are sending spam or not, the distribution is specified and then we simply spew words out independently based on the distribution. [2]

[2] You may have noticed a few years ago, when spam was still a major problem, that you might receive an email from an unknown entity with random words or phrases in it that had virtually no meaning. These emails were clearly spam but were not advertising anything. I suspect it was an attempt on the part of the spam senders, who were making millions of dollars at the time and were not stupid, to interfere with this very model and introducing noise into the Bayesian model by creating spam emails with "good" words/phrases in them.

Now we can parameterize the problem by introducing $\phi_{j|y=0} = P(w_j|y=0)$, the probability that word j in the dictionary appears in the email given that the email is not spam. Obviously, a similar definition holds for $\phi_{j|y=1}$. Now, for a particular example email i, we have

$$P(x^{(i)}|y^{(i)}) = \prod_{j=1}^{d} (\phi_{j|y=0}^{x_j^{(i)}}(1-\phi_{j|y=0})^{1-x_j^{(i)}}))^{1-y^{(i)}} (\phi_{j|y=1}^{x_j^{(i)}}(1-\phi_{j|y=1})^{1-x_j(i)})^{y^{(i)}}$$

One final piece of notation: let p be the probability that a given email is spam.

Before we attempt to solve this problem, gut check yourself and figure out how you would common-sense estimate each of these parameters...

Let the calculations begin.

$$I(x, y, p, \phi) = \sum_{i=1}^{m} -\log P(x^{(i)}|y^{(i)})P(y^{(i)}) =$$

$$\sum_{i=1}^{m} -\log \prod_{j=1}^{d} (\phi_{j|y=0}^{x_j^{(i)}}(1-\phi_{j|y=0})^{1-x_j^{(i)}})(1-p))^{1-y^{(i)}} (\phi_{j|y=1}^{x_j^{(i)}}(1-\phi_{j|y=1})^{1-x_j(i)}p)^{y^{(i)}} =$$

$$\sum_{i=1}^{m}\sum_{j=1}^{d} -\log \left[(\phi_{j|y=0}^{x_j^{(i)}}(1-\phi_{j|y=0})^{1-x_j^{(i)}})(1-p))^{1-y^{(i)}} (\phi_{j|y=1}^{x_j^{(i)}}(1-\phi_{j|y=1})^{1-x_j(i)}p)^{y^{(i)}} \right] =$$

$$\sum_{i=1}^{m}\sum_{j=1}^{d} -x_j^{(i)}(1-y^{(i)}) \log \phi_{j|y=0} - (1-x_j^{(i)})(1-y^{(i)}) \log(1-\phi_{j|y=0}) - (1-y^{(i)}) \log(1-p)$$

$$-x_j^{(i)} y^{(i)} \log \phi_{j|y=1} - (1-x_j^{(i)}) y^{(i)} \log(1-\phi_{j|y=1}) - y^{(i)} \log p$$

Taking the partial derivative with respect to $\phi_{j|y=0}$, we get

$$\frac{\partial I}{\partial \phi_{j|y=0}} = \sum_{i=1}^{m} -\frac{x_j^{(i)}(1-y^{(i)})}{\phi_{j|y=0}} + \frac{(1-x_j^{(i)})(1-y^{(i)})}{1-\phi_{j|y=0}} = 0 \Rightarrow$$

$$\sum_{i=1}^{m} x_j^{(i)}(1-y^{(i)})(1-\phi_{j|y=0}) = \sum_{i=1}^{m} (1-x_j^{(i)})(1-y^{(i)})\phi_{j|y=0} \Rightarrow$$

$$\sum_{i=1}^{m} x_j^{(i)}(1-y^{(i)}) = \sum_{i=1}^{m} (1-y^{(i)})\phi_{j|y=0} \Rightarrow$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^{m} x_j^{(i)}(1-y^{(i)})}{\sum_{i=1}^{m}(1-y^{(i)})}$$

Notice that this expression is equal to the total number of occurrences of word j in all non-spam examples divided by the total number of non-spam examples.

This is exactly what common sense says the answer should be. Taking the partial derivative with respect to $\phi_{j|y=1}$, we get

$$\frac{\partial I}{\partial \phi_{j|y=1}} = \sum_{i=1}^{m} -\frac{x_j^{(i)} y^{(i)}}{\phi_{j|y=1}} + \frac{(1 - x_j^{(i)}) y^{(i)}}{1 - \phi_{j|y=1}} = 0 \Rightarrow$$

$$\sum_{i=1}^{m} x_j^{(i)} y^{(i)} (1 - \phi_{j|y=1}) = \sum_{i=1}^{m} (1 - x_j^{(i)}) y^{(i)} \phi_{j|y=1} \Rightarrow$$

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{m} x_j^{(i)} y^{(i)}}{\sum_{i=1}^{m} y^{(i)}}$$

Notice that this expression is equal to the total number of occurrences of word j in all spam examples divided by the total number of spam examples. Again, common sense. Taking the partial dereivative with respect to p, we get

$$\frac{\partial I}{\partial p} = \sum_{i=1}^{m} \sum_{j=1}^{d} \frac{1 - y^{(i)}}{1 - p} - \frac{y^{(i)}}{p} = 0 \Rightarrow$$

$$\sum_{i=1}^{m} (1 - y^{(i)}) p = \sum_{i=1}^{m} y^{(i)} (1 - p) \Rightarrow p = \frac{\sum_{i=1}^{m} y^{(i)}}{m}$$

Finally, this implies that p is equal to the total number of spam emails divided by the total number of emails.

23 Markov Processes

Assume that you are observing a process with *transition matrix* as follows:

$$\begin{pmatrix} .5 & 0 & .5 \\ .2 & .6 & .2 \\ 1 & 0 & 0 \end{pmatrix}$$

If you are in state 1, where are you going to arrive in one time step and with what probabilities? If you are currently in state 3, where do you arrive next time? Once you leave state 2, can you ever return?

If you start in a particular state, say state 1, then the probabilities that you arrive in each of the three possible states are computed as follows:

$$(1, 0, 0) \begin{pmatrix} .5 & 0 & .5 \\ .2 & .6 & .2 \\ 1 & 0 & 0 \end{pmatrix} = (.5, 0, .5)$$

exactly as expected. Now assume that you take 2 steps starting from state 1:

$$(.5, 0, .5) \begin{pmatrix} .5 & 0 & .5 \\ .2 & .6 & .2 \\ 1 & 0 & 0 \end{pmatrix} = (\frac{3}{4}, 0, \frac{1}{4})$$

Now 3 steps:

$$(.75, 0, .25)\begin{pmatrix} .5 & 0 & .5 \\ .2 & .6 & .2 \\ 1 & 0 & 0 \end{pmatrix} = (\frac{5}{8}, 0, \frac{3}{8})$$

...and so on.

23.1 Steady state

What happens in the long run? We want a row vector $x = (x_1, x_2, x_3)$ such that

$$x\begin{pmatrix} .5 & 0 & .5 \\ .2 & .6 & .2 \\ 1 & 0 & 0 \end{pmatrix} = x$$

This will indicate that the process does nothing to the limiting distribution. Of course, this yields a system of 3 equations with 3 unkowns:

$$.5x_1 + .2x_2 + x_3 = x_1$$

$$.6x_2 = x_2$$

$$.5x_1 + .2x_2 = x_3$$

which yields the solution $x_2 = 0$ and $x_1 = 2x_3$. Recalling that $x_1 + x_2 + x_3 = 1$, we get that $x_1 = \frac{2}{3}, x_3 = \frac{1}{3}$.

23.2 Eigensystems

Let's assume that we want a simple formula to tell us where the system is at any given point in time. If we assume that we start in the state $s = (s_1, s_2, s_3)$, then what is your probability distribution after n steps of the process?

To determine this, we need to find the eigenvalues of the transition matrix M. These are the values λ such that for nonzero x, $xM = \lambda x \Rightarrow x(M - \lambda I) = 0 \Rightarrow det(M - \lambda I) = 0$. So we get

$$det(\begin{pmatrix} .5 - \lambda & 0 & .5 \\ .2 & .6 - \lambda & .2 \\ 1 & 0 & 0 - \lambda \end{pmatrix}) = 0 \Rightarrow$$

$$-\lambda^3 + \frac{11}{10}\lambda^2 + \frac{\lambda}{5} - \frac{3}{10} = 0 \Rightarrow$$

$$\lambda_1 = 1, \lambda_2 = \frac{3}{5}, \lambda_3 = -\frac{1}{2}$$

Now, we need to determine the x values that correspond with these λs.

1. $\lambda_1 = 1 \Rightarrow xM = x \Rightarrow$

$$\frac{1}{2}x_1 + \frac{1}{5}x_2 + x_3 = x_1, \frac{3}{5}x_2 = x_2, \frac{1}{2}x_1 + \frac{1}{5}x_2 = x_3 \Rightarrow$$

$$x_1 = 2x_3, x_2 = 0 \Rightarrow$$

Normalizing so that $x_1 + x_2 + x_3 = 1 \Rightarrow x = (\frac{2}{3}, 0, \frac{1}{3})$. This happens to be the same as the steady state distribution.

2. $\lambda_2 = \frac{3}{5} \Rightarrow xM = \frac{3}{5}x \Rightarrow$

$$x_1 = \frac{8}{3}x_3, x_2 = -\frac{11}{3}x_3$$

It is interesting that this case is impossible to normalize... the sum of the variables must always equal 0. We can, for the sake of concreteness, assume that $x_3 = 3 \Rightarrow x = (8, -11, 3)$.

3. $\lambda_3 = -\frac{1}{2} \Rightarrow xM = -\frac{1}{2}x \Rightarrow$

$$x_1 = -x_3, x_2 = 0$$

Once again, it is not possible to normalize. Letting $x_3 = 1 \Rightarrow x = (-1, 0, 1)$.

Let us define the matrix X to be x_1 along the first row, x_2 along the second row, and x_3 along the third row. So

$$X \equiv \begin{pmatrix} \frac{2}{3} & 0 & \frac{1}{3} \\ 8 & -11 & 3 \\ -1 & 0 & 1 \end{pmatrix}$$

Let us define

$$E \equiv \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$$

It is clear from all that we have done above that

$$XM = EX \Rightarrow M = X^{-1}EX$$

Taking n steps from a start state s implies that we are calculating

$$sM^n = s(X^{-1}EX)^n = sX^{-1}E^nX$$

For concreteness, we have that

$$X^{-1} = \begin{pmatrix} 1 & 0 & -\frac{1}{3} \\ 1 & -\frac{1}{11} & -\frac{2}{33} \\ 1 & 0 & \frac{2}{3} \end{pmatrix} \Rightarrow$$

$$X^{-1}E^nX = \begin{pmatrix} \frac{2}{3} + \frac{1}{3}(-\frac{1}{2})^n & 0 & \frac{1}{3} - \frac{1}{3}(-\frac{1}{2})^n \\ \frac{2}{3} - \frac{8}{11}(\frac{3}{5})^n + \frac{2}{33}(-\frac{1}{2})^n & (\frac{3}{5})^n & \frac{1}{3} - \frac{2}{33}(-\frac{1}{2})^n - \frac{3}{11}(\frac{3}{5})^n \\ \frac{2}{3} - \frac{2}{3}(-\frac{1}{2})^n & 0 & \frac{1}{3} + \frac{2}{3}(-\frac{1}{2})^n \end{pmatrix}$$

To check that this is correct, plug in $n = 0$ and make sure that the correct matrix pops out (HINT: the identity). Plug in $n = 1$ and make sure that the correct matrix pops out (HINT: M). So, in general, if you start in some state $s = (s_1, s_2, s_3)$, then you can find the probability that you are in any given state *exactly* at any given time n by computing $sX^{-1}E^nX$.

To find the limiting distribution, let's find out what happens when $n \to \infty$...

$$\lim_{n\to\infty} sX^{-1}E^nX = s \begin{pmatrix} \frac{2}{3} & 0 & \frac{1}{3} \\ \frac{2}{3} & 0 & \frac{1}{3} \\ \frac{2}{3} & 0 & \frac{1}{3} \end{pmatrix} =$$

$$(\frac{2}{3}(s_1 + s_2 + s_3), 0, \frac{1}{3}(s_1 + s_2 + s_3)) = (\frac{2}{3}, 0, \frac{1}{3})$$

So, eventually, independent of where you started, you will eventually spend two-thirds of your time in state 1 and one-third of your time in state 3.

23.3 Markov decision processes

Claim: $a + ar + ar^2 + \ldots + ar^{n-1} = a\frac{r^n-1}{r-1}$

23.4 Setup

Let us assume that you have a wife/girlfriend (we will analyze each separately below) and are choosing a gift for some holiday. She has 2 states, happy and not happy, and those states depend on your choice of gift/her mood/other unknown quantities. State 1 will refer to your wife/girlfriend as happy and state 2 will refer to your wife/girlfriend as unhappy. The transition matrix for her moods are as follows.

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{2}{5} & \frac{3}{5} \end{pmatrix}$$

Let the value p_{ij} refer to the i, j entry of this matrix (i.e. the probability that you transition from state i to state j).

Assume that if she is happy twice in a row, you receive a reward of 9. If she is unhappy twice in a row, you receive a reward of -7. If she transitions from one mood to another, you receive a reward of 3. This allows us to set up a *reward matrix* as follows.

$$\begin{pmatrix} 9 & 3 \\ 3 & -7 \end{pmatrix}$$

Note that the payoffs are paid based on the *transitions* and not solely on the state she arrives in. Define q_i to be the expected reward for the next transition

if the current state is i (with the obvious definition for the vector q). Then we have the following simple equation for q_i: $q_i = \sum_j p_{ij} r_{ij}$. In our case, we get

$$q = \begin{pmatrix} \frac{1}{2}9 + \frac{1}{2}3 \\ \frac{2}{5}3 + \frac{3}{5}(-7) \end{pmatrix} = \begin{pmatrix} 6 \\ -3 \end{pmatrix}$$

We define the quantity $v_i(n)$ as your expected total rewards after n transitions assuming you start from state i. The vector $v(n)$ is therefore defined to be the *expected total rewards* after n transitions. Note that $v(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

23.5 Infinite horizon (wife)

Assume that you are married and that you intend to have an infinite number of interactions. What is your average reward per interaction in the long term? Asymptotically, how much can you expect to make as a function of n, the number of interactions, and the initial state of your wife?

We can note the following recurrence for $v(n)$ with the knowledge that this is a Markovian process. This basically states that you get the reward for the first step and then the reward for the subsequent $n-1$ steps.

$$v_i(n) = \sum_j p_{ij}[r_{ij} + v_j(n-1)] = \sum_j p_{ij} r_{ij} + \sum_j p_{ij} v_j(n-1) =$$

$$q_i + \sum_j p_{ij} v_j(n-1)$$

In vector form, this works out to

$$v(n) = q + Pv(n-1)$$

We get the following sequence for $v(i)$.

$$v(1) = \begin{pmatrix} 6 \\ -3 \end{pmatrix}, v(2) = \begin{pmatrix} \frac{15}{2} \\ -\frac{12}{5} \end{pmatrix} = \begin{pmatrix} 7.5 \\ -2.4 \end{pmatrix},$$

$$v(3) = \begin{pmatrix} \frac{171}{20} \\ -\frac{36}{25} \end{pmatrix} = \begin{pmatrix} 8.55 \\ -1.44 \end{pmatrix}, v(4) = \begin{pmatrix} \frac{1911}{200} \\ -\frac{111}{250} \end{pmatrix} = \begin{pmatrix} 9.555 \\ -0.444 \end{pmatrix},$$

$$v(5) = \begin{pmatrix} \frac{21111}{2000} \\ \frac{1389}{2500} \end{pmatrix} = \begin{pmatrix} 10.5555 \\ 0.5556 \end{pmatrix}, v(6) = \begin{pmatrix} \frac{231111}{20000} \\ \frac{38889}{25000} \end{pmatrix} = \begin{pmatrix} 11.5556 \\ 1.55556 \end{pmatrix},$$

$$v(7) = \begin{pmatrix} \frac{2511111}{200000} \\ \frac{638889}{250000} \end{pmatrix} = \begin{pmatrix} 12.5556 \\ 2.55556 \end{pmatrix}, v(8) = \begin{pmatrix} \frac{27111111}{2000000} \\ \frac{8888889}{2500000} \end{pmatrix} = \begin{pmatrix} 13.5556 \\ 3.55556 \end{pmatrix}, \ldots$$

Using the recurrence, we get that

$$v(1) = q + Pv(0), v(2) = q + P(q + Pv(0)) = q + Pq + P^2 v(0), v(3) = q + Pq + P^2 q + P^3 v(0), \ldots \Rightarrow$$

$$v(n) = \sum_{i=0}^{n-1} P^i q + P^n v(0) = \sum_{i=0}^{n-1} P^i q$$

To find a closed form for P^i, we need to find its eigenvalues and eigenvectors. We claim that the eigenvalues are 1 and $\frac{1}{10}$ and the respective eigenvectors are $\begin{pmatrix}1\\1\end{pmatrix}$ and $\begin{pmatrix}-\frac{5}{4}\\1\end{pmatrix}$. Plugging this into the formula, we get that

$$P^i = \begin{pmatrix}\frac{1}{2} & \frac{1}{2}\\ \frac{2}{5} & \frac{3}{5}\end{pmatrix}^i = \begin{pmatrix}1 & -\frac{5}{4}\\ 1 & 1\end{pmatrix}\begin{pmatrix}1^i & 0\\ 0 & (\frac{1}{10})^i\end{pmatrix}\begin{pmatrix}1 & -\frac{5}{4}\\ 1 & 1\end{pmatrix}^{-1} \Rightarrow$$

Plugging into the formula for $v(n)$, we get that

$$v(n) = \sum_{i=0}^{n-1} P^i q =$$

$$\begin{pmatrix}1 & -\frac{5}{4}\\ 1 & 1\end{pmatrix}\left[\sum_{i=0}^{n-1}\begin{pmatrix}1^i & 0\\ 0 & (\frac{1}{10})^i\end{pmatrix}\right]\begin{pmatrix}1 & -\frac{5}{4}\\ 1 & 1\end{pmatrix}^{-1}\begin{pmatrix}6\\-3\end{pmatrix} =$$

$$\begin{pmatrix}1 & -\frac{5}{4}\\ 1 & 1\end{pmatrix}\begin{pmatrix}n & 0\\ 0 & \frac{10(10^n-1)}{9(10)^n}\end{pmatrix}\begin{pmatrix}1 & -\frac{5}{4}\\ 1 & 1\end{pmatrix}^{-1}\begin{pmatrix}6\\-3\end{pmatrix} =$$

$$\begin{pmatrix}\frac{50}{9}(1-(\frac{1}{10})^n)+n\\ -\frac{40}{9}(1-(\frac{1}{10})^n)+n\end{pmatrix}$$

Note then that on average, you make $\lim_{n\to\infty}\frac{v(n)}{n} = 1$ per move regardless of where she starts. Asymptotically, the score you can expect is $n+\frac{50}{9}$ if she starts out happy and $n-\frac{40}{9}$ if she starts out unhappy.

23.6 Discounting (Girlfriend)

Now assume that you may not be with this particular girl forever. In this case, you are looking to determine her value to you with a built-in "breakup" possibility built in to the math. Assume that there is some constant $0 < \beta < 1$ such that the probability that you will break up with her after n transitions is $1-\beta^n$; in other words, after the first transition, given that you are still going out with the girl, the probability that you will buy her another gift is β, independent of the number of interactions thus far. (Note that this implies that you will definitely make at least one transition.)

There are several consequences to this type of *discounting*.

- Beginning interactions are counted as far more important than later ones.
- The total reward will always converge (assuming the reward matrix is made up of constants).
- Rather than a difficult recursive relation, we can compute the infinite sum directly.

Note that

$$v = \left[\sum_{j=0}^{\infty}(\beta P)^j\right] q = (I - \beta P)^{-1} q \Rightarrow$$

$$v = \begin{pmatrix} \frac{60-51\beta}{10-11\beta+\beta^2} \\ \frac{39\beta-30}{10-11\beta+\beta^2} \end{pmatrix}$$

In this way, you can assess this interaction's lifetime value quickly and easily, depending on your choice of β.

24 Markov policies

Assume that you are in the same situation as before, trying to keep your wife/girlfriend in a good mood. However, at any given point in time, you may elect to buy an extremely expensive gift. This gift will set you back some cash but will increase the probability that she becomes happy assuming she is currently unhappy. In short, you may elect to change the transition matrix from $\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{2}{5} & \frac{3}{5} \end{pmatrix}$ to $\begin{pmatrix} \frac{4}{5} & \frac{1}{5} \\ \frac{7}{10} & \frac{3}{10} \end{pmatrix}$; however, this will change your reward matrix from $\begin{pmatrix} 9 & 3 \\ 3 & -7 \end{pmatrix}$ to $\begin{pmatrix} 6 & 0 \\ 0 & -10 \end{pmatrix}$. There is one catch, you need to make the decisions about what you intend to buy right now in order to budget for the big gifts. (If we were allowed to know exactly what state she was in at any given time, this would be a much easier problem. Why?) What is the optimal *policy* for buying expensive gifts as opposed to the normal flowers? Let policy 1 refer to buying flowers and policy 2 refer to buying diamonds. Note that even if we assume that we are only going to make n transitions, the total number of policies that we could use is 2^n; searching through all of these possibilities to find the best one is not feasible.

Define $p_{ij}^{(k)}$ to be the transition matrix probabilities if we use policy k and similarly let $r_{ij}^{(k)}$ represent rewards if we use policy k. Define $q_i^{(k)}$ to be the expected reward for the next transition if the current state is i and policy k is used. Let $d_i(n)$ be the optimal policy to use assuming that the system is in state i with $n \geq 1$ transitions remaining.

Again, let $v_i^*(n)$ be the total expected reward that the system obtains in n transitions assuming it starts from state i and an optimal policy is used. Obviously, $v^*(n)$ represents the associated vector. As usual, $v^*(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. We can then write down the following recursive relation based on our previous discussions.

$$v_i^*(n+1) = \max_k \sum_j p_{ij}^{(k)}[r_{ij}^{(k)} + v_j^*(n)] = \max_k [q_i^{(k)} + \sum_j p_{ij}^{(k)} v_j^*(n)]$$

So we can work from the bottom up. For our example, we know from

our previous lecture that $q^{(1)} = \begin{pmatrix} 6 \\ -3 \end{pmatrix}$. It is easy to work out that $q^{(2)} = \begin{pmatrix} \frac{4}{5}6 + \frac{1}{5}0 \\ \frac{7}{10}0 + \frac{3}{10}(-10) \end{pmatrix} = \begin{pmatrix} \frac{24}{5} \\ -3 \end{pmatrix}$.

$$v^*(1) = \begin{pmatrix} \max\{6, \frac{24}{5}\} \\ \max\{-3, -3\} \end{pmatrix} \Rightarrow$$

$$v^*(1) = \begin{pmatrix} 6 \\ -3 \end{pmatrix}, d(1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$v^*(2) = \begin{pmatrix} \max\{6 + \frac{1}{2}(6) + \frac{1}{2}(-3), \frac{24}{5} + \frac{4}{5}(6) + \frac{1}{5}(-3)\} \\ \max\{-3 + \frac{2}{5}(6) + \frac{3}{5}(-3), -3 + \frac{7}{10}(6) + \frac{3}{10}(-3)\} \end{pmatrix} \Rightarrow$$

$$v^*(2) = \begin{pmatrix} 9 \\ \frac{3}{10} \end{pmatrix}, d(2) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$v^*(3) = \begin{pmatrix} \max\{6 + \frac{1}{2}(9) + \frac{1}{2}(\frac{3}{10}), \frac{24}{5} + \frac{4}{5}(9) + \frac{1}{5}(\frac{3}{10})\} \\ \max\{-3 + \frac{2}{5}(9) + \frac{3}{5}(\frac{3}{10}), -3 + \frac{7}{10}(9) + \frac{3}{10}(\frac{3}{10})\} \end{pmatrix} \Rightarrow$$

$$v^*(3) = \begin{pmatrix} 12.06 \\ 3.39 \end{pmatrix}, d(3) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$v^*(4) = \begin{pmatrix} \max\{6 + \frac{1}{2}(12.06) + \frac{1}{2}(3.39), \frac{24}{5} + \frac{4}{5}(12.06) + \frac{1}{5}(3.39)\} \\ \max\{-3 + \frac{2}{5}(12.06) + \frac{3}{5}(3.39), -3 + \frac{7}{10}(12.06) + \frac{3}{10}(3.39)\} \end{pmatrix} \Rightarrow$$

$$v^*(4) = \begin{pmatrix} 15.126 \\ 6.459 \end{pmatrix}, d(4) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$d(5) = d(6) = d(7) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$v^*(5) = \begin{pmatrix} 18.1926 \\ 9.5259 \end{pmatrix}, v^*(6) = \begin{pmatrix} 21.2593 \\ 12.5926 \end{pmatrix}, v^*(7) = \begin{pmatrix} 24.3259 \\ 15.6593 \end{pmatrix}$$

and so on. From this point, it should be clear that $\forall i > 7, d(i) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$. We can interpret these results as follows: Up until the very last transition, no matter what state she is in, it is worth it to buy the diamonds. During the last transition, it is not worth it.

What happens if the diamonds are double the price, 6 instead of 3? The reward matrix for buying the expensive gift becomes $\begin{pmatrix} 3 & -3 \\ -3 & -13 \end{pmatrix}$. The first values of v^* and d are as follows.

$$v^*(1) = \begin{pmatrix} 6 \\ -3 \end{pmatrix}, d(1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$v^*(2) = \begin{pmatrix} 7.5 \\ -2.4 \end{pmatrix}, d(2) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$v^*(3) = \begin{pmatrix} 8.55 \\ -1.44 \end{pmatrix}, d(3) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$v^*(4) = \begin{pmatrix} 9.555 \\ -0.444 \end{pmatrix}, d(4) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$v^*(5) = \begin{pmatrix} 10.5555 \\ 0.5556 \end{pmatrix}, d(5) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$v^*(6) = \begin{pmatrix} 11.5556 \\ 1.55556 \end{pmatrix}, d(6) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$v^*(7) = \begin{pmatrix} 12.5556 \\ 2.55556 \end{pmatrix}, d(7) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

In this case, you will never buy a diamond. They are too expensive to be worth it at this price.

What happens if the diamonds are slightly more expensive? Rather than costing only 3, let's assume that the diamonds cost 4.45. Does this change anything? The reward matrix for buying the expensive gift becomes $\begin{pmatrix} 4.55 & -1.45 \\ -1.45 & -11.45 \end{pmatrix}$. The first values of v^* and d are as follows.

$$v^*(1) = \begin{pmatrix} 6 \\ -3 \end{pmatrix}, d(1) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$v^*(2) = \begin{pmatrix} 7.55 \\ -1.15 \end{pmatrix}, d(2) = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$v^*(3) = \begin{pmatrix} 9.2 \\ 0.49 \end{pmatrix}, d(3) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$v^*(4) = \begin{pmatrix} 10.845 \\ 2.137 \end{pmatrix}, d(4) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$v^*(5) = \begin{pmatrix} 12.491 \\ 3.7826 \end{pmatrix}, d(5) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$v^*(6) = \begin{pmatrix} 14.1368 \\ 5.42848 \end{pmatrix}, d(6) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

$$v^*(7) = \begin{pmatrix} 15.7826 \\ 7.0743 \end{pmatrix}, d(7) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

We can interpret these results as follows: While she is happy, there is no need to buy her a gift *unless it is the second to last transition* in which case it is worth it for the possibility that she will make the happy-happy transition during the last transition. If she becomes unhappy, then it is worth it to pay for the diamonds *unless it is the final transition* in which case it is not worth it for the same reasons as before.

25 Adversarial search

SEARCH trees: If you're a chess person, it may interest you to know that this is how the machine Deep Blue (the first computer ever to beat a Grand Master, Garry Kasparov, at chess) is programmed to work. Kasparov's ratings achievements include being rated world No.1 according to Elo rating almost continuously from 1986 until his retirement in 2005 and holding the all-time highest rating of 2851. He was the world number-one ranked player for 255 months, by far the most of all-time and nearly three times as long as his closest rival, Anatoly Karpov. He also holds records for consecutive tournament victories and Chess Oscars.

The two played in 1996 and Kasparov won, but in 1997...

- 1997 Game 1, Kasparov vs. Deep Blue (Winner: Kasparov) The 1997 rematch began with the King's Indian Attack, which led Kasparov to victory in 45 moves.

- 1997 Game 2, Deep Blue vs. Kasparov (Winner: Deep Blue) In this game Kasparov accused IBM of cheating, a claim repeated in the documentary Game Over: Kasparov and the Machine. Kasparov eventually resigned, although post-game analysis indicates that the game could have been drawn. At the time it was reported that Kasparov missed certain moves, Black can force a draw by perpetual check (threefold repetition). His friends told him so the next morning. They suggested a sequence of moves to result in a quick ending. It was believed that the game became subject to a forced draw instead of a probable win for Deep Blue. Regarding the end of game 2, chess journalist Mig Greengard in the Game Over film states, "It turns out, that the position in, here at the end is actually a draw, and that, one of Deep Blue's final moves was a terrible error, because Deep Blue has two choices here. It can move its king here or move its king over here. It picked the wrong place to step." Another in that film, four-time US champion Yasser Seirawan, then concludes that, "The computer had left its king a little un-defended. And Garry could have threatened a perpetual check, not a win but a perpetual check."

  However, it was discovered in 2007 by Internet analysis that after these moves, there is no perpetual check and the attack continues. After lengthy analysis, all of Deep Blue's moves make sense and it has never been shown that it made a mistake.

- 1997 Game 3, Kasparov vs. Deep Blue (Draw) The third game was interesting because Kasparov chose to use an irregular opening. He believed that by playing an esoteric opening, the computer would get out of its opening book and play the opening worse than it would have done using the book. Although this is nowadays a common tactic, it was a relatively new idea at the time. Despite this anti-computer tactic, the game was eventually drawn.

- 1997 Game 4, Deep Blue vs. Kasparov (Draw) In this game Kasparov played the Caro-Kann Defence. Later on he had time problems and had to play in a hurry, as both players had two hours for the first 40 moves and Kasparov was approaching his time limit. The sub-optimal moves he played in a hurry may have cost him the victory. Drawn.
- 1997 Game 5, Kasparov vs. Deep Blue (Draw) In this game, the King's Indian Attack opening was played. Like in the previous game, Deep Blue played a brilliant endgame that secured a draw, when it was looking like Kasparov would win.
- 1997 Game 6, Deep Blue vs. Kasparov (Deep Blue) Before the sixth match the overall score was even: $2\frac{1}{2}$ games each. As in Game 4, Kasparov played the Caro-Kann Defence. He then allowed Deep Blue to commit a knight sacrifice which wrecked his defences and forced him to resign in fewer than twenty moves.

Rules

1. Assume that you are playing a two-player game with an adversary.

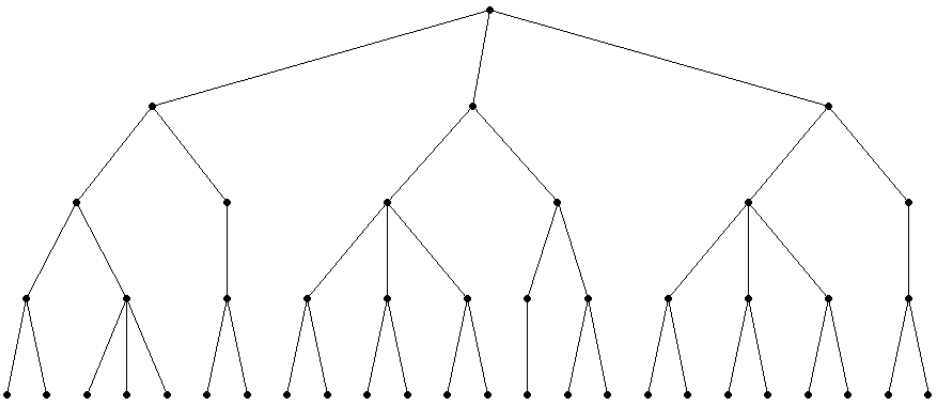
2. We are going to assume that this adversary is not stupid and therefore will always make the move that is best for him at any given time.

3. We assume that you alternate moves with the adversary.

4. The object of the game is to maximize the total number of "points" that you get. (Of course, your adversary's objective is to minimize your total number of points so as to maximize his own. We assume that we are playing a zero-sum game.)

5. We assume that you get to move first. (It won't change the analysis of the game any if the other player gets to move first.)

6. Finally, we assume that you only have a certain small fixed amount of time to decide which move you want to do before you have to make your choice.



Starting from the root and working right and down, let the sequence of node scores be as follows:

0
3, -1, 2
1, 4, 2, -3, -5, -2
0, -6, 5, 9, -2, -8, 13, -1, 2, 8, -5, 1
-4, -2, 4, 2, 7, -2, 9, 1, -3, -4, 1, 0, -7, 3, -6, 5, 4, -1, -8, 9, -2, 6, 0, -9

Assuming both players can see the whole game tree, what is the sequence of moves that occurs during this game? 0,3,4,5,-2

What if the adversary is allowed to go first? 0,-1,-3,13,3

You move first. Your computer is powerful enough to see the whole game tree. Your adversary's computer is only powerful enough to see two levels down from his current position in the game tree. You and your adversary both are aware of your computer's superiority. 0,3,1,-6,2

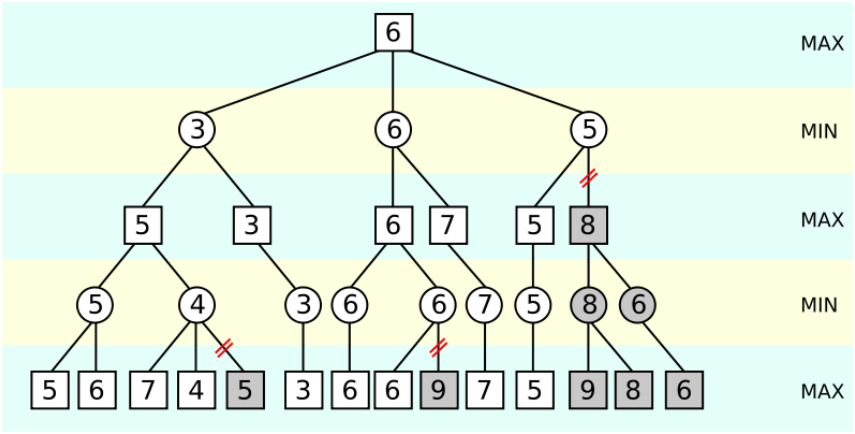
You move first. Both computers are powerful enough to see the entire game board. You think that his computer is weak and can only see one move ahead. What happens? 0,-1,2,9,-3

26 Alpha-beta pruning

The algorithm maintains two values, α and β, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially α is $-\infty$ and β is $+\infty$. As the recursion progresses the "window" becomes smaller.

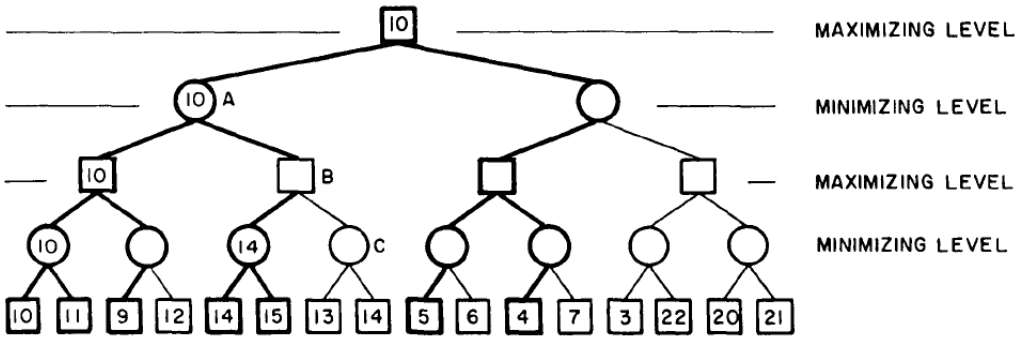
It is important to note that α can only change when it is the maximizer's turn to move and β can only change when it is the minimizer's turn to move. These values change when we move upward only. When we move downward, they do not change.

When $\beta \leq \alpha$, it means that the current position cannot be the result of best play by both players and hence need not be explored further: to see why, assume that α is some fixed value and that β suddenly changes so that $\beta \leq \alpha$. This implies that the maximizer can, by choosing to follow some node other than the current one achieve a value of α, whereas if the game reaches the current node, then the minimizer is guaranteed to get you down to β. Will the maximizer ever let the game reach the current node? Of course not. He will chose to move elsewhere so that this node need not be explored futher. The same argument works in reverse for when β is fixed and α changes.



26.1 Additional facts about alpha-beta pruning

Fig. 1. A Binary Game Tree of Depth 4 Traversed from Left-to-Right by the Alpha–Beta Procedure.



Notice that performing alpha-beta pruning on this tree from left to right yields many prunes whereas performing the search from right to left yields none. All 16 terminal nodes will be examined with a right to left search whereas a left to right search will only examine 7!

We will now examine some theoretical results: Define $N_{n,d}$ to be the average number of terminal nodes examined by the alpha-beta pruning algorithm in a uniform game tree of degree n and depth d for which the terminal values are drawn at random from a continuous distribution. We define the *branching factor* of the alpha-beta pruning procedure to be

$$\mathcal{R}_{\alpha-\beta} \equiv \lim_{d\to\infty} (N_{n,d})^{1/d}$$

To get an idea for why this is a good definition, consider the number of terminal nodes in a tree with uniform degree n and depth d; this value is n^d (take the dth root to get the value n). In particular, the branching factor is the factor that

multiplies the number of terminal nodes in a tree of depth d to get the number of terminal nodes in a tree of depth $d+1$.

Early empirical studies of the alpha-beta branching factor (Fuller et al. 1973) yielded $\mathcal{R}_{\alpha-\beta} \approx n^{0.72}$. Knuth and Moore (1975) proved that the branching factor of a simplified model yields $O(\frac{n}{\log n})$ and speculated that this result would also hold for $\mathcal{R}_{\alpha-\beta}$. Baudet (1978) proved that the branching factor could be estimated by

$$\frac{\eta_n}{1-\eta_n} \leq \mathcal{R}_{\alpha-\beta} \leq \sqrt{M_n}$$

where η_n is the positive root of $x^n + x - 1 = 0$ and M_n is the maximal value of the polynomial $P(x) = (\frac{1-x^n}{1-x})(\frac{1-(1-x^n)^n}{x^n})$ in the range $0 \leq x \leq 1$. Pearl (1980) and Baudet (1981) together showed a lower bound of $\frac{\eta_n}{1-\eta_n}$ for the branching factor of *any* game tree search algorithm. Finally, Pearl (1982) showed that $\mathcal{R}_{\alpha-\beta} = \frac{\eta_n}{1-\eta_n}$ and alpha-beta is thus optimal in the sense described.

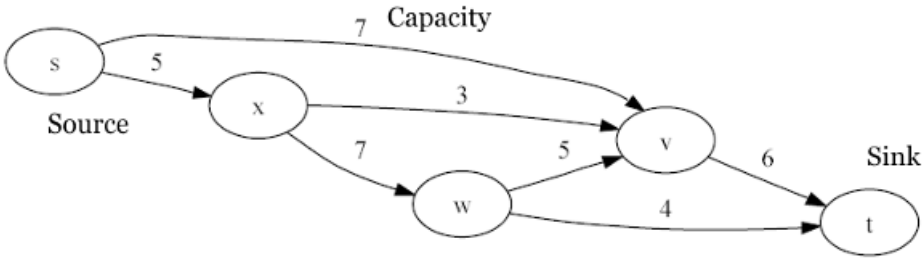
Pearl (1982) states: "The asymptotic behavior of $\mathcal{R}_{\alpha-\beta}$ is $O(n/\log n)$ as predicted by Knuth's analysis. However, for moderate values of n ($n \leq 1000$), $\frac{\eta_n}{1-\eta_n}$ is fitted much better by the formula $(0.925)n^{0.747}$ which vindicates the simulation results of Fuller et al. (1973). This approximation offers a more meaningful appreciation of the pruning power of the $\alpha-\beta$ algorithm. Roughly speaking, a fraction of only $(0.925)n^{0.747}/n \approx n^{-1/4}$ of the legal moves will be explored by $\alpha-\beta$. Alternatively, for a given search time allotment, the $\alpha-\beta$ pruning allows the search depth to be increased by a factor $\log n/\log \mathcal{R}_{\alpha-\beta} \approx 4/3$ over that of an exhaustive minimax search."

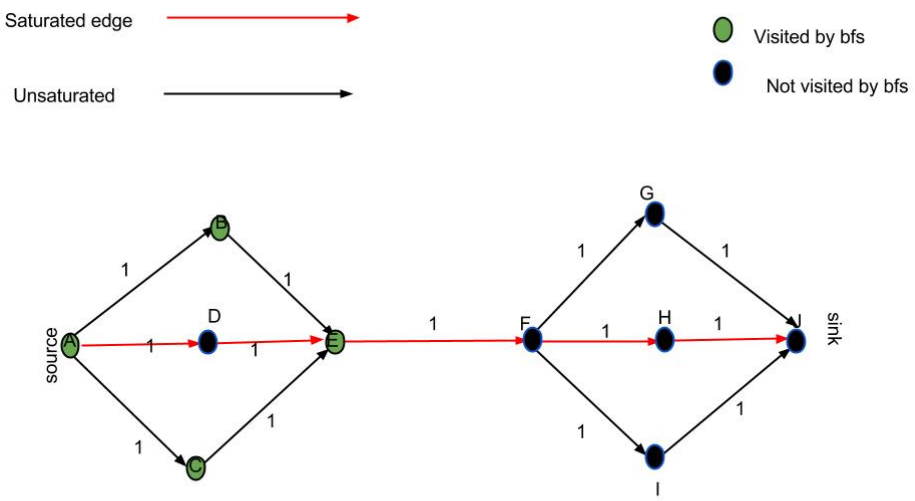
Judea Pearl is the winner of the 2012 Turing award and father of Daniel Pearl.

27 Image Segmentation

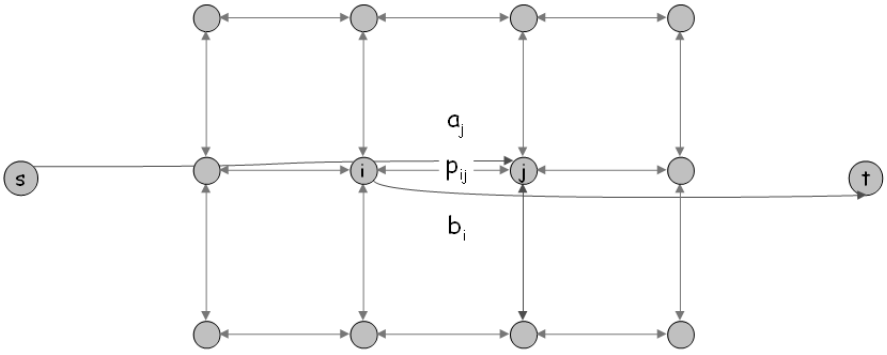
27.1 A minimum cut algorithm

Perform a maximum flow computation on the graph. Now perform a depth-first search on the residual network. Any edges from a visited vertex to a nonvisited vertex are in the minimum cut.





27.2 The algorithm



Consider the problem of taking an image and determine which shapes correspond to the foreground and background. Pictures are split into millions of pixels; which pixels are foreground?

Let V be the set of pixels of a picture. Let E be a bidirectional edge between two pixels if and only if they are adjacent in the picture. For each pixel i, we let $a_i \geq 0$ be the reward for choosing pixel i to be foreground and we let $b_i \geq 0$

be the reward for choosing pixel i to be background. We let $p_{ij} = p_{ji} \geq 0$ be the separation penalty for labeling one of i and j as foreground and the other background. These a, b, p parameters are all determined by some image processing algorithm and can be assumed to be input parameters (i.e. given quantities) to our problem.

Let the set of foreground pixels that we choose be F and let the background be B.

There are two goals in this problem:

1. Accuracy: If $a_i > b_i$, we would obviously prefer to label pixel i as foreground (and vice versa).
2. Smoothness: If neighbors of pixel i are labeled foreground, we should be inclined to label i as foreground as well (and vice versa).

Therefore, the function that we will attempt to maximize is (assuming $F \cup B = V$ and $F \cap B = \emptyset$)

$$\sum_{i \in F} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E : |F \cap \{i,j\}| = 1} p_{ij}$$

Note that this is equivalent to minimizing

$$\sum_{i \in B} a_i + \sum_{j \in F} b_j + \sum_{(i,j) \in E : |F \cap \{i,j\}| = 1} p_{ij}$$

Let the network be described as follows: To each vertex/pixel i, let there be an edge with weight a_i from the source to i and from each vertex/pixel j, let there be an edge with weight b_j to the sink. If pixel i is adjacent to pixel j, let there be an edge in both directions of weight p_{ij}. Consider a maximum flow in this network. What does the minimum cut look like?

Claim: Given a maximum flow, let X be the vertices on the source side of the corresponding minimum cut and Y be the vertices of the sink side. We claim that if $x \in X$ and $y \in Y$, then the following edges must be in the minimum cut: a_y, b_x, and p_{xy} (from x to y).

Proof: If $x \in X$, then x can be reached by some additional flow from the source. If b_x was not in the cut, then this additional flow could be led directly to the sink, contradicting the maximality of the flow. If $y \in Y$, then there is additional capacity in some path from y to the sink. If a_y was not in the cut, then flow could be directed through this edge to y and then onwards to the sink, contradicting the maximality of the flow again. Finally, let us consider the edge labeled p_{xy} from x to y. Because $x \in X$, there is additional capacity from the source to x and because $y \in Y$, there is additional capacity from y to the sink. If p_{xy} was not in the cut, then our flow again would not be maximal. □

Claim: The minimum cut comprises exactly those edges a_y, b_x, p_{xy} (from x to y) such that $x \in X, y \in Y$.

Proof: We already know that the minimum cut contains at least these edges. It will suffice to show that there are no additional edges that will ever be included

in a minimum cut. To show this, consider all paths out from the source assuming only the described edges are cut: they can only reach vertices in X. From there, there is no way to get to the sink, only to other vertices in X. Hence, these edges qualify as a true cut, separating source from sink. Adding additional edges would only increase the cut value. □

Note now that the edges that are in the minimum cut comprise exactly the edges we want to minimize in our problem. So a simple algorithm for computing the foreground/background is as follows: Create the network as above and compute the maximum flow. Let the vertices reachable by the source be $X = F$, the foreground, and let the vertices that can reach the sink be $Y = B$, the background.

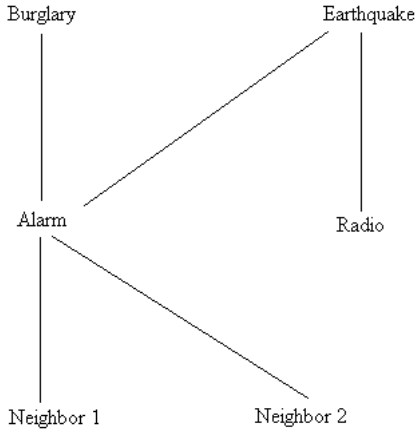
28 Introduction to Bayesian networks

Invented by Judea Pearl (teaches at UCLA, Turing award winner in 2012, father of Daniel Pearl) in 1985 to put knowledge into more compact and more intuitive form for both humans and computers. "Lumierre" was a 1993 attempt by Microsoft to use Bayesian networks to anticipate the needs of its users; the effort eventually culminated in the emergence of the "assistants" you see in Microsoft's late 90's office products (e.g. that damned talking paperclip, the search dog, etc.).

Bayes Rule: $P(A|B) = \frac{P(A \cap B)}{P(B)}$

Throughout the rest of this lecture, we use the Einstein summation convention: If there is an uninstantiated variable in the expression, we sum over all possible instatiation of that variable.

Pearl: In LA, burglary and earthquakes are both common. If an earthquake occurs, it may be mentioned on the radio. If the alarm in your house goes off, it can be caused by either a burglary or an earthquake. You have two neighbors who have promised to call you if they hear the alarm (but most alarms in the neighborhood sound alike). Assume that Neighbor 1 calls and reports that he thinks he might have heard your alarm go off; you only want to return home if there was a burglary. What is the probability that there has been a burglary at your house given that Neighbor 1 has called your cell phone? Let us name the various events as follows: E,B,R,1,2,A. We are asking for $P(B = true|1 = true) = \frac{P(B=true \cap 1=true)}{P(1=true)}$. In order to compute this with no further information, we would have to remember an extremely large conditional probability table: $\frac{P(B=true,E,R,1=true,2,A)}{P(B,E,R,2,A,1=true)}$ (summing over all of the noninstantiated variables). In fact, in general, if you have n binary variables and you want to store a joint distribution, you will, in general, need to store 2^n distinct real numbers.



Assume that the following are known:

$$P(B = true) = .001, P(E = true) = .002$$

$$P(A = true|B = true, E = true) = .999, P(A = true|B = false, E = true) = .05$$

$$P(A = true|B = true, E = false) = .71, P(A = true|B = false, E = false) = .01$$

$$P(R = true|E = true) = 0.75, P(R = true|E = false) = 0.001$$

$$P(1 = true|A = true) = .95, P(1 = true|A = false) = .1$$

$$P(2 = true|A = true) = .99, P(2 = true|A = false) = .3$$

(Notice that this is far fewer than 2^n and, in most practical applications, is a pretty small polynoimal in n.)

Now consider what happens if we use Bayes Rule:

$$P(X_1 \cap X_2 \cap \ldots \cap X_n) = P(X_1) \cdot P(X_2 \cap \ldots X_n|X_1) =$$

$$P(X_1) \cdot P(X_2|X_1) \cdot P(X_3 \cap \ldots \cap X_n) = \ldots = \prod_{i=1}^{n} P(X_i|X_1, X_2, \ldots, X_{i-1})$$

You still have to remember a ton of information.

First, set up the Bayesian network for this system. Note that it is a DAG (as are all Bayesian networks). Note that each node only depends on the state of its parent. Then

$$P(B = true \cap 1 = true) = P(B = true, 1 = true, E, R, 2, A) =$$

$$P(B = true) \cdot P(E|B = true) \cdot P(A|B = true, E) \cdot P(R|B = true, A, E)$$

$$\cdot P(1 = true|B = true, A, E, R) \cdot P(2|1 = true, B = true, A, E, R) =$$

$$P(B = true) \cdot P(E) \cdot P(A|B = true, E) \cdot P(R|E) \cdot P(1 = true|A) \cdot P(2|A) =$$

because summing over R and 2 leads to 1

$$P(B = true) \cdot P(E) \cdot P(A|B = true, E) \cdot P(1 = true|A)$$

Now we have only 4 issues to worry about $E = true/false$ and $A = true/false$. We need to sum over all possibilities (and there are only 4!):

$$[E, A] = [true, true] + [true, false] + [false, true] + [false, false]$$

.001·.002·.999·.95+.001·.002·.001·.1+.001·.998·.71·.95+.001·.998·.29·.1 = 0.000703991

(This information set is much smaller and more intuitive.) Using the same reasoning,

$$P(1 = true) = P(B) \cdot P(E) \cdot P(A|B, E) \cdot P(R|E) \cdot P(1 = true|A) \cdot P(2|A) =$$

$$P(B) \cdot P(E) \cdot P(A|B, E) \cdot P(1 = true|A)$$

We will follow the following sum format.

$[B, E, A] = [true, true, true]+[true, true, false]+[true, false, true]+[true, false, false]+$

$[false, true, true]+[false, true, false]+[false, false, true]+[false, false, false]$

$.001 \cdot .002 \cdot .999 \cdot .95 + .001 \cdot .002 \cdot .001 \cdot .1 + .001 \cdot .998 \cdot .71 \cdot .95 + .001 \cdot .998 \cdot .29 \cdot .1 +$

.999·.002·.05·.95+.999·.002·.95·.1+.999·.998·.01·.95+.999·.998·.99·.1 = 0.109163

So $P(B = true|1 = true) = 0.00644897$.

29 Markov blankets

When are X and Y independent in a Bayesian network? There are 4 possibilities:

1. If Z is a random variable on a direct path from Y to X, then if Z is measured, the information that might be contained in Y is irrelevant to X along that particular path. [For example, if you want to find out whether X =Neighbor 2 is going to call, knowing whether Y =Burglary happened is irrelevant if you know the Z =Alarm went off.] Measuring Z therefore stops information flow from Y to X.

2. If Z is a random variable on a direct path from X to Y, then if Z is measured, then no additional information can be gained by measuring Y along that particular path. [For example, if you are interested in knowing whether there was a Burglary (X), then knowing whether Neighbor 1 called your house (Y) is not helpful if you already know whether the Alarm (Z) went off.] Measuring Z therefore steps information flow from Y to X in this case as well.

3. Assume that Z is an ancestor of both Y and X. If Z is measured, then no information can be passed along the (undirected) path from Y to Z to X. [If you know that Z =Alarm went off, then X =Neighbor 1 calling and Y =Neighbor 2 calling are completely independent. On the other hand, if you have no other information, one value certainly affects the other.] Measuring Z stops information flow from Y to X.

4. Assume that Z is a descendent of both Y and X. If NO nodes along the (undirected) path from X to Z to Y are measured, then no information can be passed along the (undirected) path from X to Z to Y. [Without any other information, the events X =Earthquake and Y =Burglary are totally independent. However, if you know that Z =Alarm went off, then knowing the outcome of one event clearly influences the outcome of the other.] *Not* measuring Z stops information flow from Y to X.

If there does not exist a Z node anywhere in the network that makes it possible for information to be passed from Y to X, X is independent of Y. In this case, we say that the nodes X and Y are **d-separated**.

The **Markov blanket** of a node X is the smallest subset of the nodes of the network that need to be measured in order to make the node X independent from the rest of the nodes in the network.

Claim: The Markov blanket consists of exactly the parents, children, and parents of children of X.

Proof: Fix a node X somewhere in the network, and assume that we have *only* measured the parents, children, and children of parents of X. Can there be a node located anywhere in the network that remains unmeasured such that, if measured, can affect the distribution of node X? There are 3 cases for where such a node Y might be located.

- Y is a descendent of X. In this case, measuring the children of X cuts off information flow from Y to X.

- Y is an ancestor of X. In this case, measuring the parents of X cuts off information flow from Y to X.

- Y is not an ancestor or descendent of X. (i) There may be a common ancestor to X and Y. If such a node Z exists, then Z cannot pass information toward the node X because X's parents are measured. Thus, measuring Y would have no effect via any path through any ancestor. (ii) There may also be a common descendent from X and Y, call it Z. If Z is *not* one of X's children, then we have already cut off information flow from Z to X by measuring X's children. So assume that Z is one of X's children, and it has been measured. Then, on first glance, measuring Y will change X's distribution... However, notice that we have also measured X's children's parents; thus, either Y has already been measured or Y, an ancestor of Z, will not be able to pass information along to Z because Z's parents have all been measured.

□

30 Monte Carlo Markov Chain methods for learning with Bayesian networks

30.1 Buffon's Needle

This is a gentle introduction to the concept behind Monte Carlo methods for approximation.

Suppose we have a floor made of parallel strips of wood, each the same width w, and we drop a needle of length $d < w$ onto the floor. What is the probability that the needle will lie across a line between two strips?

Let X be the distance from the center of the dropped needle to the nearest line. Let θ be the acute positive angle that the needle makes with the line. Notice that X has an equal chance of being anything between 0 and $\frac{w}{2}$ and θ has an equal chance of being anything between 0 and $\frac{\pi}{2}$. Then, assuming that θ has been fixed, the probability that the needle lies across a line is equal to the probability that the center is within $\frac{d}{2}\sin\theta$ of the closest line. So this probability is

$$\int_0^{\frac{d}{2}\sin\theta} P(X = x)dx = \int_0^{\frac{d}{2}\sin\theta} \frac{2}{w}dx$$

So now the probability that we get a particular θ value can be factored in:

$$\int_0^{\frac{\pi}{2}} \int_0^{\frac{d}{2}\sin\theta} \frac{2}{w}dx\frac{2}{\pi}d\theta = \frac{2d}{\pi w}$$

So, given knowledge of the separation between the strips and the length of the needle, you can simply drop the needle a large number of times and count the number of times it overlaps one of the lines. Eventually, you will get an approximation for this fraction which you can then use to estimate the number π.

30.2 Gibb's sampling

Credit for inventing the Monte Carlo method often goes to Stanislaw Ulam, a Polish born mathematician who worked for John von Neumann on the United States' Manhattan Project during World War II. Ulam is primarily known for designing the hydrogen bomb with Edward Teller in 1951. He invented the Monte Carlo method in 1946 while pondering the probabilities of winning a card game of solitaire. Quoted in Eckhardt (1987), Ulam describes the incident as: "The first thoughts and attempts I made to practice [the Monte Carlo Method] were suggested by a question which occurred to me in 1946 as I was convalescing from an illness and playing solitaires. The question was what are the chances that a Canfield solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than "abstract thinking" might not be to lay it out say one hundred times and simply observe and count the number of successful plays. This was already possible to envisage with the beginning of

the new era of fast computers, and I immediately thought of problems of neutron diffusion and other questions of mathematical physics, and more generally how to change processes described by certain differential equations into an equivalent form interpretable as a succession of random operations. Later I described the idea to John von Neumann, and we began to plan actual calculations."

We will be doing Gibbs sampling. In the following analysis, technically you need to show that the Markov chain that we create is ergodic (any state can be reached from any other state). Assume that you are given a Bayesian network with nodes N and that a certain number of these nodes have been measured; we call these evidence nodes $E \subseteq N$. The goal is to get an idea of what a given probability distribution is assuming the values of the evidence nodes.

Below, we will be referring to the *state* of the Markov chain. The state will refer to the current value of the nodes $N - E$ in the Bayesian network; the evidence nodes always have the same values, namely those given by the evidence. The Gibbs sampler algorithm runs as follows:

1. Initialize the state of the nodes $N - E$ to anything valid. Call this initial state $x^{(0)}$.
2. For $t = 1, 2, \ldots$
   (a) Choose a node $n \in N - E$ uniformly at random and change n's value based on the current value of n's Markov blanket.
   (b) Update the probability you are interested in based on the new state.

Let's consider the burglary/earthquake model. Assume that you want to determine the probability that a burglary happened assuming that Neighbor 1 called. In this case, the evidence node is the Neighbor 1 node and the state of that node is always true. Recall that there are 6 nodes: (E,B,R,1,2,A); all nodes in our example are binary. We can initialize the state to (true,false,false,true,false,true) where every entry was randomly chosen except for the 4th (namely, the 1 entry). In this case, E is true so we are 1 for 1. If we choose a nonevidence node at random, say B, then we can, given all the values of all the other nodes, determine a new value for B. How do we do so?

For notational convenience, let make some definitions. First, let the parents of n be P. Let the children of n be C and let the parents of children of n that are not n itself be PC. Then we have that (denoting probability by p)

$$p(n|N-\{n\}) = p(n|MB(n)) = p(n|P,C,PC) = p([n|P]|C,PC) = \frac{p(C,PC|[n|P])p(n|P)}{p(C,PC)} \propto$$

$$p(n|P)p(C,PC|[n|P]) = p(n|P)p(C|PC,n,P)p(PC|n,P)$$

Note that $p(C|PC,n,P) = p(C|PC,n)$ because the children of n are independent of n's parents given the value of the children's parents. We claim that the parents of the children of n and n itself are d-separated, given that n's parents have been measured. If n's parents have been measured, then the only way for information to flow between n and another node n' is if there exists a

common descendent that has been measured. Clearly, that is not possible; thus $p(PC|n,P) = p(PC|P)$ which does not involve n.

$$p(n|P)p(C|PC,n,P)P(PC|n,P) = p(n|P)p(C|PC,n)p(PC|P) \propto p(n|P)p(C|PC,n)$$

and because the children of n are all independent of each other given their own parents, we have the following proportionality.

$$p(n|N - \{n\}) \propto p(n|P) \prod_{c \in C} p(c|\text{parents}(c))$$

30.3 Calculations and implementation

To determine $P(B)$ given the state (E,B,R,1,2,A)=(true,false,false,true,false,true),

$$P(B = true) \propto P(B = true) \cdot P(A = true|B = true, E = true) = .001 \cdot .999 = .00999$$

$$P(B = false) \propto P(B = false) \cdot P(A = true|B = false, E = true) = .999 \cdot .05 = 0.04995$$

Thus, $P(B = true) = 0.166667$ and $P(B = false) = 0.833333$. Choose a new value for B based on this distribution and continue.

When this was plugged into the Gibbs sampler for one million state transitions, the total count for the number of states where B was true was 6274. If you recall, the calculation we did yesterday predicted a probability of 0.00644897 which corresponds pretty well with the expreimental results.

31 Towards SVMs

31.1 Coordinate Ascent (Special case)

You use simulated annealing when you've got a function that you know nothing about with few dimensions and genetic algorithms when there is an obvious way to combine two good points to get a potentially better one, but consider the case when the job is to optimize a function with multiple dimensions (though within certain well-defined boundaries) and no obvious way to combine points.

Consider the following function where all x_i's are variables. Our goal will be to maximize this function[3] assuming that all of the r_i and a_i values are randomly chosen and the value of n is a large number and $C > 0$ is some constant. (This looks like a random problem but it will be of great use to us later on.)

$$f(x) = \sum_{i=1}^{n} x_i - \sum_{i,j=1}^{n} r_{ij} x_i x_j$$

$$0 \leq x_i \leq C, \sum_{i=1}^{n} a_i x_i = 0$$

[3]Recall that maximizing a function is equivalent to minimizing the negative of the function.

Because there are 100 dimensions to this problem, simulated annealing is not likely to make much progress, and because it is difficult to determine how to combine two good answers to make a third good answer, a genetic algorithm might not accomplish much more. Note also that we cannot simply take partial derivates of all the variables and solve the resulting equations, as the resulting optima may lie outside the prescribed hypercube and will almost certainly not satisfy the final constraint.

Instead, we choose a point at random to begin that we know is inside the hypercube and satsfies the constraint, say $\forall i, x_i = 0$. Then we choose all but two variables to remain constant and allow the remaining two to vary.

For example, let the problem be posed as follows:

$$f(x) = x_1 + x_2 + x_3 - 3x_1x_2 + 2x_1x_3 + 6x_2x_3$$
$$0 \leq x_1, x_2, x_3 \leq 5, x_1 - x_2 + 2x_3 = 0$$

Start by letting the solution be $x_1 = x_2 = x_3 = 0$.

1. Decide (essentially at random) to fix the variable x_2 and let x_1 and x_3 vary. Then (recalling that $x_2 = 0$ currently)

   $$f(x)_{x_2=0} = x_1 + x_3 + 2x_1x_3$$
   $$0 \leq x_1, x_3 \leq 5, x_1 + 2x_3 = 0$$

   If we now notice that the final equation implies that $x_1 = -2x_3$, the first equation is a quadratic in x_3. Notice that the maximum value for a quadratic must occur either at the unique maximum found by simply taking the derivative and setting it equal to 0 or at one of the endpoints. In this case, we get that the maximum occurs at $x_3 = -\frac{1}{8}$ which is outside of our bounds. Testing the four bounds $(x_1, x_3) = (0,0), (0,5), (5,0), (5,5)$, we get that the maximum of the function $f(x)_{x_2=0}$ occurs when $(x_1, x_3) = (5,5)$. Thus, our current solution is

   $$(x_1, x_2, x_3) = (5, 0, 5)$$

2. Decide to fix the variable x_1 and let x_2 and x_3 vary. Then (recalling that $x_1 = 5$ currently)

   $$f(x)_{x_1=5} = 5 - 14x_2 + 11x_3 + 6x_2x_3$$
   $$0 \leq x_2, x_3 \leq 5, 5 - x_2 + 2x_3 = 0$$

   Notice that the final equation implies that $x_2 = 5 + 2x_3$. Then, plugging in for x_2, we get that

   $$f(x)_{x_1=5, x_2=5+2x_3} = 12x_3^2 + 13x_3 - 65$$

   Because the coefficient of the quadratic term is positive, we know that the maximum will occur on one of the extremal points of the hypercube. Thus, we again test $(x_2, x_3) = (0,0), (0,5), (5,0), (5,5)$... The maximum of the function $f(x)_{x_1=5}$ occurs at $x_2 = x_3 = 5$.

3. Decide to fix the variable x_3. Then (recalling that $x_3 = 5$ currently)

$$f(x)_{x_3=5} = 5 + 11x_1 + 31x_2 - 3x_1x_2$$

$$0 \leq x_1, x_2 \leq 5, x_1 - x_2 + 10 = 0$$

Letting $x_1 = x_2 - 10$, we get

$$f(x)_{x_3=5, x_1=x_2-10} = -3x_2^2 + 72x_2 - 105$$

which has its minimum at $x_2 = 9$ which is outside our range. We test the endpoints $(x_1, x_2) = (0,0), (0,5), (5,0), (5,5)...$ The maximum of the function $f(x)_{x_3=5}$ occurs at $x_1 = 0, x_2 = 5$. Thus, our solution is currently

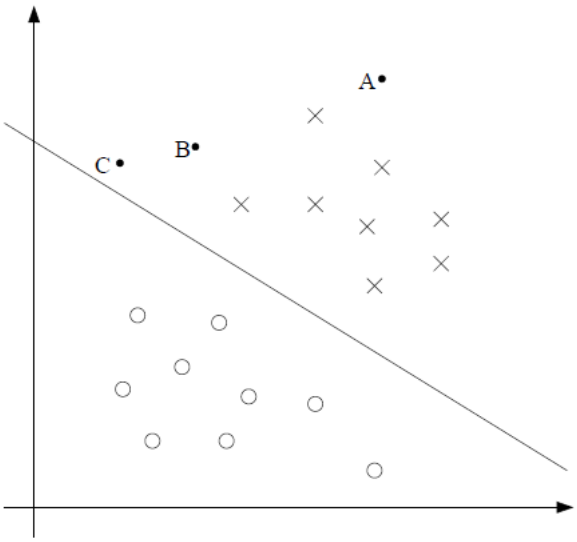
$$x_1 = 0, x_2 = 5, x_3 = 5$$

Further iterations of the algorithm will not yield an improvement and this is the maximum.

31.2 Derivation

Assume that there are m column vectors $x^{(1)}, \ldots, x^{(m)}$ that need to be classified into positive and negative categories. For each vector $x^{(i)}$, we are provided with an "answer" $y^{(i)} \in \{-1, 1\}$. We will assume that there exists some line that goes through the origin that separates the points. Our goal will be to find the "best" line that separates the points, a linear separator. We will assume that there exists some parameter vector θ such that the feature vector is linearly related to the answer; in other words, we are assuming a linear model. Once we choose our parameter row vector θ, if we are given a new item with feature vector x, we determine its y answer by evaluating the quantity θx; if the quantity if positive, then we declare that $y = 1$ and if the quantity is negative, then $y = -1$. Note that this is, for the most part, exactly the same setup as for perceptrons (except that we are not yet allowing the constant term into our model).

We will be removing these assumptions one by one as we develop our mathematical model.



1. Obviously, some lines are better than others. Note that it is possible to draw a line through the point C in the above picture that separates the points, but it would feel less satisfying than the line that is drawn. Why is that? We are looking for the line that splits the points into two sets in such a way that the minimum distance between the closest point to the separator on either side is maximized. Putting this mathematically, we are looking for a parameter vector θ that maximizes the constant c such that $\forall i, y^{(i)}\theta x^{(i)} \geq c$. (Notice that we are more confident with our prediction if the value of c is large.) Thus, our problem becomes

$$\max c \text{ subject to } \forall i, y^{(i)}\theta x^{(i)} \geq c$$

2. Note that we can take every constraint above and divide through by c, yielding

$$\max c \text{ subject to } \forall i, y^{(i)}\frac{\theta}{c}x^{(i)} \geq 1$$

Now we can note that if we are maximizing the value of the constant c, it is equivalent to minimizing the norm of θ. Minimizing $||\theta||$ is the same as minimizing the quantity $\frac{1}{2}||\theta||^2$. Thus, we can rewrite our problem as follows

$$\min \frac{1}{2}||\theta||^2 \text{ subject to } \forall i, y^{(i)}\theta x^{(i)} \geq 1$$

3. What if we are not guaranteed that the separator goes through the origin. We would like to allow ourselves freedom of adjusting the position of the line should we need to. Thus, we allow ourselves a constant b such that the problem now becomes

$$\min \frac{1}{2}||\theta||^2 \text{ subject to } \forall i, y^{(i)}(\theta x^{(i)} + b) \geq 1$$

This effectly puts back the constant term from the perceptron model.

4. Clearly, not all points will be able to lie perfectly on one side of the line or the other. Thus, we introduce the constants p_i (penalty constants) and choose some constant C so that the problem becomes

$$\min \frac{1}{2}||\theta||^2 + C\sum_{i=1}^{m} p_i \text{ subject to } \forall i, y^{(i)}(\theta x^{(i)} + b) \geq 1 - p_i, p_i \geq 0$$

In other words, if your ideal separator does not meet the threshold of 1 by some amount p_i, your objective function is penalized by an amount Cp_i. This is the full formulation of the problem.

32 Maximal linear separators and support vector machines

32.1 Solving the problem

Let

$$g(\theta, p) = \frac{1}{2}\theta\theta^T + C\sum_{i=1}^{m} p_i$$

Introduce the Lagrangian as follows.

$$f(\theta, b, p, \alpha, \beta) = g(\theta, p) + \sum_{i=1}^{m} \alpha_i[1 - p_i - \left(y^{(i)}(\theta x^{(i)} + b)\right)] + \sum_{i=1}^{m} \beta_i p_i$$

where α and β represent Lagrange multiplier vectors. Consider the value of $\max_{\alpha \geq 0, \beta \leq 0} f(\theta, b, p, \alpha, \beta)$. Note that if any of the conditions are violated for some values of θ, b, p and we allow arbitrary values in the ranges $\alpha \geq 0$ and $\beta \leq 0$, then $\max_{\alpha \geq 0, \beta \leq 0} f(\theta, b, p, \alpha, \beta)$ becomes ∞. On the other hand, if the constraints are all satisfied, then we get back the value $g(\theta, b, p)$, what we want. Our goal is therefore to evaluate

$$\min_{\theta, b, p} \max_{\alpha \geq 0, \beta \leq 0} f(\theta, b, p, \alpha, \beta)$$

We will invoke the KKT conditions (which we do not know anything about nor care) and claim that this is equal to

$$\max_{\alpha \geq 0, \beta \leq 0} \min_{\theta, b, p} f(\theta, b, p, \alpha, \beta)$$

Taking the derivative (gradient) with respect to θ, we get

$$\nabla_\theta f = \theta^T - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0$$

which implies that

$$\theta = \sum_{i=1}^{m} \alpha_i y^{(i)} (x^{(i)})^T$$

Taking the derivative (gradient) with respect to b, we get

$$\nabla_b f = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

Taking the derivative with respect to p_i, we get

$$\frac{\partial f}{\partial p_i} = C - \alpha_i + \beta_i = 0 \Rightarrow \beta_i = \alpha_i - C$$

Plugging back in and simplifying slightly based on the p_i derivative result, we get

$$f(\theta, b, p, \alpha, \beta) = \frac{1}{2}\theta\theta^T + \sum_{i=1}^{m} \alpha_i - \sum_{i=1}^{m} \alpha_i (y^{(i)}(\theta x^{(i)} + b))$$

Using the results from the derivative with respect to b, we get

$$f(\theta, b, p, \alpha, \beta) = \frac{1}{2}\theta\theta^T + \sum_{i=1}^{m} \alpha_i - \sum_{i=1}^{m} \alpha_i y^{(i)} \theta x^{(i)}$$

Using the results from the derivative with respect to θ, we get

$$f(\theta, b, p, \alpha, \beta) = \sum_{i=1}^{m} \alpha_i + \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i y^{(i)} (x^{(i)})^T \alpha_j y^{(j)} x^{(j)} - \sum_{i,j=1}^{m} \alpha_i y^{(i)} x^{(i)} \alpha_j y^{(j)} (x^{(j)})^T \Rightarrow$$

$$f(\theta, b, p, \alpha, \beta) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i y^{(i)} (x^{(i)})^T \alpha_j y^{(j)} x^{(j)} =$$

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i y^{(i)} \alpha_j y^{(j)} < x^{(i)}, x^{(j)} >$$

Now we need to minimize this function with respect to α and β with the restrictions that $\alpha_i \geq 0$ and $\beta_i \leq 0$ *and* $\beta_i = \alpha_i - C$. These final two conditions lead to the condition that $0 \leq \alpha_i \leq C$. We are left with the following problem (based on the final form of the function, the restrictions we just found, and the derivative with respect to b after having eliminated θ and β)

$$\text{Minimize } \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i y^{(i)} \alpha_j y^{(j)} < x^{(i)}, x^{(j)} > \text{ subject to}$$

$$0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

This is the final form of the support vector machine problem.

There are a few interesting things to note about this solution.

1. First, note this problem was built for coordinate ascent.

2. Notice that the problem itself requires solving for the values of α. Notice that, by construction of the problem, α_i should hopefully be equal to 0 most of the time. In fact, if the inequality constraints for which the α_i was constructed is not an equality, then the corresponding α_i must turn out to be 0. (i.e. $1-p_i \neq (y^{(i)}(\theta x^{(i)}+b)) \Rightarrow \alpha_i = 0 \Rightarrow \beta_i = -C \Rightarrow p_i = 0$) The values of $x^{(i)}$ for which the constraints exactly meet the equality are called *support vectors*. They "support" the maximal separator.

3. The most important thing to notice is that the only place the x's appear in the newly formed problem, they appear within an inner product. Additionally, if we are given a new value of x, then to make a prediction, we calculate (using the result from the derivative with respect to θ)

$$\theta x + b = b + \sum_{i=1}^{m} \alpha_i y^{(i)} (x^{(i)})^T x =$$

$$b + \sum_{i=1}^{m} \alpha_i y^{(i)} < x^{(i)}, x >$$

Because $x^{(i)}$ ONLY appears in inners products in this problem and solution, there is no need to necessarily compute the $x^{(i)}$ vectors themselves nor store them in memory. As an example, let's consider the case, where we have a vector of features $(x_1, \ldots, x_n)^T$ but we suspect that the result of our computation should depend on the products betweens these features in addition to the features themselves (i.e. the real number of features should be of the order $\Theta(n^2)$). Rather than computing the straight inner product $< x, z >= x^T z$, replace it everywhere with the inner product $(x^T z+1)^2$. Note that, asymptotically, this takes the same amount of time to compute. However, we have the following equations.

$$(x^T z + 1)^2 = \sum_{i,j=1}^{n} x_i x_j z_i z_j + 2 \sum_{i=1}^{n} x_i z_i + 1 =$$

$$\sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n} (\sqrt{2} x_i)(\sqrt{2} z_i) + 1$$

We have just magically transformed the feature vector $\phi : (x_1, \ldots, x_n) \mapsto (x_1 x_1, x_1 x_2, \ldots, x_1 x_n, x_2 x_1, \ldots, x_2 x_n, \ldots, x_n x_n, \sqrt{2} x_1, \sqrt{2} x_2, \ldots, \sqrt{2} x_n, 1)$ because

$$(x^T z + 1)^2 =< \phi(x), \phi(z) >$$

We will not prove the following theorem. What it states is that almost all reasonable definitions of the inner product correspond to some mapping of the feature vector.

Theorem(Mercer): For any feature vectors $x^{(1)}, \ldots, x^{(m)}$ and a mapping ϕ, define the *kernel* to be $K(x^{(i)}, x^{(j)}) =< \phi(x^{(i)}), \phi(x^{(j)}) >$. Define the kernel matrix K such that $K_{ij} = K(x^{(i)}, x^{(j)})$. In order for a given matrix K to be the kernel matrix for some mapping ϕ, K needs to be (i) symmetric (obviously) and (ii) positive semi-definite (i.e. $\forall z, z^T K z \geq 0$).

- An interesting example of an inner product is $K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}$. This is called the *Gaussian kernel* and is a perfectly valid measure of the similarity between x and y.
- Another interesting example of a kernel is, given some k, one that maps a string of characters x to the number of occurrences of each length-k substring in x. If we consider strings of English alphabet characters, then $|\phi(x)| = 26^k$. However, consider computing $< \phi(x), \phi(z) >= \phi(x)^T \phi(z)$ for two distinct strings x and z; in other words, we want to compute the sum of the products of every length k substring that appears some number of times in both. A very simple naive algorithm is to take every substring of length k in x, look for the number of times it appears in z. This can be run in time $O(k|x||z|)$. Notice that the feature vector itself is enormous whereas the computation time for the inner product is bilinear in the length of the strings themselves.