CS221 Fall 2014 Homework 1

SUNet ID: thomasme
Name: Thomas Mendoza
Collaborators: Steven Samson

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

(a) In order to minimize the function $f(x) = \frac{1}{2}\sum_{i=1}^{n} w_i(x - b_i)^2$ we must establish the derivative which is:

$$f'(x) = \sum_{i=1}^{n} w_i(x - b_i)$$

and solve for when it is equal to zero since the minimum is characterized by the tangent of zero slope for quadratic functions. Choosing an x such that the summation in the derivative is as close to zero as possible means that the quantity $(x - b_i)$ must be either as close to zero as possible or cause terms to sum to zero (e.g. $(x - b_m) + (x - b_n) = 0$) which can be achieved by choosing x to be the mean of the b_i terms.

(b) The relationship between the functions:

$$f(x) = \max_{a\in\{1,-1\}} \sum_{j=1}^{d} ax_j \text{ and } g(x) = \sum_{j=1}^{d} \max_{a\in\{1,-1\}} ax_j$$

is that $f(x) \leq g(x)$. The function $f(x)$ computes the max of two sums, one in which all a are positive and the other when all a are negative over the sum. The function $g(x)$ on the other hand computes the max on every term of the sum, $\max(\{x_j, -x_j\})$, which will yield a positive number for every term in the sum despite the sign on x. Since the sum in $f(x)$ is a sum of positive and negative numbers it will be smaller than $g(x)$, a sum of all positive terms, unless all x are positive in which case $f(x)$ and $g(x)$ will be equal.

(c) For n rolls of a fair, six-sided dice, the expected value of the counter tracking dots $d \geq 4$ can be expressed by $E = p(d \geq 4)n$. The value of $p(d \geq 4)$ is $\frac{1}{2}$ since there are 3 of 6 sides where $d \geq 4$. Thus the expected value, E, is $\frac{n}{2}$. The dice rolls *must* be independent to calculate the expected value, otherwise $p(d \geq 4)$ for any given roll would need to be computed using the probabilities of outcomes of previous rolls.

(d) In order to compute the value p that maximizes $L(p)$ we must take the derivative and solve for when it is equal to zero. First, before taking the derivative of $L(p) = p^3(1-p)^2$ it is useful to apply natural log to simplify the expression:

$$ln(L(p)) = ln(p^3(1 - p)^2)$$

$$ln(L(p)) = 3ln(p) + 2ln(1-p)$$

Differentiating and setting the left hand side to zero:

$$0 = \frac{3}{p} - \frac{2}{1-p}$$

Which, when simplified, yields $p = 0.6$. This answer is reasonable because, in order to achieve heads, H, 3 times in a sequence of 5, the optimal bias on the coin would require H to appear 60% of the time.

(e) To find $\nabla f(w)$:

$$\nabla f(w) = \frac{\partial}{\partial w}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^T w - b_j^T w)^2 + \lambda\|w\|_2^2\right)$$

which by linearity can be written as:

$$\nabla f(w) = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{\partial}{\partial w}(a_i^T w - b_j^T w)^2 + \lambda\frac{\partial}{\partial w}\|w\|_2^2$$

differentiating the terms under the sum yields:

$$\nabla f(w) = 2\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^T w - b_j^T w)(a_i^T - b_j^T) + \lambda\frac{\partial}{\partial w}\|w\|_2^2$$

derivative of the 2-norm is $\frac{w}{\|w\|_2^2}$ which will arise as part of the chain rule, so differentiating the term multiplying λ will result in:

$$2\|w\|_2^2\left(\frac{w}{\|w\|_2^2}\right) = 2w$$

rendering the gradient:

$$\nabla f(w) = 2\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^T w - b_j^T w)(a_i^T - b_j^T) + 2\lambda w$$

Problem 2

(a) For each word in the text of n words, there are four possible tags. The number of possible sequences is given by taking the number tag of choices x for each word in the text and multiplying them, otherwise written:

$$\prod_{i=1}^{n} x_i$$

In this case there are four possible tags ($x = 4$) yielding 4^n possible sequences.

(b) In order to find the asymptotic complexity for possible faces using two eyes and one mouth, its instructive to make certain considerations about the problem constraints. Since there are no size or position constraints on the eyes and mouth, they each share the same possible arrangements, A, meaning the total possible arrangements would be $3A$. We're only concerned with asymptotic complexity, so we can focus on finding A for a single feature. The A arrangements can be thought of by choosing two horizontal and two vertical lines to form the perimeter. In the nxn grid there are $n+1$ total vertical and horizontal lines to choose from. Using the previous statements we can describe these choices using combinatorics, namely:

$$\binom{n+1}{2}\binom{n+1}{2} = \left(\frac{(n+1)!}{2!(n-1)!}\right)^2 = \frac{n^2(n+1)^2}{4}$$

This means that the asymptotic time to compute all possible A is $O(n^4)$

(c) The journey from city i to city j can be depicted as a directed graph where edge values denote the cost of going forward from one city to the next (vertices in the graph). This problem is most efficiently solved by Djikstra's algorithm for computing the least-cost path between two points in a graph. Djikstra's algorithm in effect computes a memoized version of the recurrence:

$$f(j) = \min_{1 \le i < j}[c(i,j) + f(i)]$$

since it does not revisit any previously considered vertices. In general, the asymptotic run-time is $O(|V|^2)$ where V represents the number of vertices. For sparse graphs however, this can be reduced to $O(|E| + |V|\log(|V|))$.

(d) To alter the function

$$f(w) = \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^T w - b_j^T w)^2 + \lambda\|w\|_2^2$$

to allow for preprocessing in $O(nd^2)$ time and future computation in just $O(d^2)$ time, we can factor out w from inside the sum:

$$f(w) = \sum_{i=1}^{n}\sum_{j=1}^{n}(w(a_i^T - b_j^T))^2 + \lambda\|w\|_2^2$$

for which w can be removed from squaring by recognizing that its square involves its transpose:

$$f(w) = w^T\left(\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^T - b_j^T)^2\right)w + \lambda\|w\|_2^2$$

This is the expression we're looking for. The sums contained in parenthesis are completely independent of w and can therefore be computed ahead of time–a process which computes the square in d^2 time for the quantity $(a_i^T - b_j^T)^2$ and does so n times, yielding the $O(nd^2)$ compute time. Further computation multiplies this pre-computed value with updated w and does so in $O(d^2)$ time.