CS221 Fall 2015 Homework [number]

SUNet ID: ziyuanc
Name: Ziyuan Chen
Collaborators: N/A

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) $\frac{df(x)}{dx} = \sum_{i=1}^{n}(\omega_i x - \omega_i b_i) = 0$
$x = \frac{\sum_{i=1}^{n}\omega_1 b_i}{\sum_{i=1}^{n}\omega_i}$ minimizes the function.
If some ω_i's are negative, then second derivative of $f(x)$ could be negative. Therefore, the value x obtained from the expression above could be the maximum instead of the minimum.

(b) We need to refactor the expressions
$f(x) = \max_{a\in\{1,-1\}} a\sum_{j=1}^{d} x_j = |\sum_{j=1}^{d} x_j|$
$g(x) = \sum_{j=1}^{d}\max_{a\in\{1,-1\}} ax_j = \sum_{j=1}^{d}|x_j|$
According to the triangle inequality, $g(x) \geqslant f(x)$

(c) Use the geometric distribution
The expected number of rolls before stopping is $\frac{1}{\frac{1}{3}} = 3$. For each roll, the chance of getting r points is $\frac{1}{6}$. Thus the expected number of points is $3 \times \frac{1}{6}r = \frac{1}{2}r$

(d) Take the derivative of $logL(p)$:
$\frac{dlogL(p)}{dp} = \frac{d(2log(p)+3log(1-p))}{dp} = 0$
So $\frac{2}{p} = \frac{3}{1-p} = \frac{2}{p} - \frac{3}{1-p} = 0$ $p = \frac{2}{5}$
If we take the second derivative, then we have $\frac{-2}{p^2} - \frac{3}{(1-p)^2}$ which is -20.8 at the p we computed above. Thus it is a maximum point.

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

Problem 2

(a) A rectangle has (n-a+1)(n-b+1) placements if it is a×b. Therefore the number of possible placements of a rectangle is $O(n^2)$.
Because both a and b are in the range [1,n], a rectangle has $O(n^2)$ different possible dimensions.
Therefore for one rectangle there are $O(n^4)$ ways to place it.
For 5 rectangles, it should be in the order of $(n^4)^5 = n^{20}$. Thus it's $O(n^{20})$.

(b) The algorithm looks like:

```
min_cost={}
def 2b(i):
    if i==n:
        return 0
    if min_cost[i] already exists:
        return min_cost[i]
    else:
        answer=MAX_INT
        for j in range(i+1,n+1):
            answer=min(answer,c(i,j)+2b(j))
        min_cost[i]=answer
        return answer
```

This algorithm has a time complexity of $O(n^2)$ with dynamic programming.

(c) If we draw out the rectangle and fill out each block with the number of ways to get there from the upper-left, then it is quite obvious that there are $\frac{(2n-2)!}{((n-1)!)^2}$ ways.

(d) Since $\mathbf{w}$ is independent of the summations, just factor it out. We get:
$f(\mathbf{w}) = \mathbf{w}^2 \sum_{i=1}^{n} \sum_{j=1}^{n} (a_i^\top - b_j^\top)^2 + \lambda \sum_{i=1}^{d} \mathbf{w}_i$
In preprocessing we can calculate
$sum_{i=1}^{n} \sum_{j=1}^{n} (a_i^\top - b_j^\top)^2$ first, which takes $O(nd^2)$ time. Then for any given $\mathbf{w}$ we can simply compute $\sum_{i=1}^{d} \mathbf{w}_i$, which takes $O(d^2)$ time.