Algorithms

MinSeok Song

Algorithmic paradigm

1 Divide and Conquer

Theorem 1 (Master method)**.** *Let* $T(n) = aT\left(\frac{n}{b}\right) + f(n)$*. Then:*

1. *If* $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ *for some* $\epsilon > 0$*, then* $T(n) = \Theta(n^{\log_b a})$.
2. *If*
   - $f(n) = \Theta\left(n^{\log_b a}\right)$*, then* $T(n) = \Theta\left(n^{\log_b a} \log n\right)$.
   - $f(n) = \Theta\left(n^{\log_b a} \log^k n\right)$*, then* $T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right)$.
3. *If* $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ *for some* $\epsilon > 0$ *and* $af\left(\frac{n}{b}\right) \leqslant cf(n)$ *for some* $c < 1$*, then* $T(n) = \Theta(f(n))$.

- We use substitution method to guess the solution or use induction to prove it formally.
- Recursion-tree method to discover the right guess (or also can be used for proof).
- We use Master Method to prove formally.
- This method doesn't always work: $T(n) = 4T(n/2) + n^2/\log n$ (in-between cases of 1 and 2).

Example 1.
- Merge Sort: $T(n) = 2 \cdot T(n/2) + \Theta(n)$.
- Binary Search: $T(n) = T(n/2) + \Theta(1)$.
- Insertion Sort: $T(n) = \Theta(n)$ (best case), $T(n) = \Theta(n^2)$ (worst case)
  - Iterate from 0 to $n-1$, swap if in reverse order.
- Matrix multiplication (Strassen'60): $T(n) = 7T(n/2) + \Theta(n^2)$ (scheme with 7 recursive multiplications of $n/2 \times n/2$ submatrices).
- Quickselect (finding i'th smallest element): $T(n) = \max\{T(|\text{left part}|),$ $T(|\text{right part}|)\} + \Theta(n)$
  - Best Case: $\Theta(n)$
  - Worst Case: $\Theta(n^2)$: This happens when we have sorted A. We may improve this to get $\Theta(n)$. How? First method is to use random pivot. Second method, which is deterministic, would be to choose a pivot element judiciously. Group n elements into $\lceil n/5 \rceil$ groups and calculate median for each group. Then we wil get $T(n) \geqslant T(\lceil n/5 \rceil) + T(\frac{7n}{10} + 2) + \Theta(n)$, which gives $T(n) = \Theta(n)$.
  - Typical case: $\Theta(n)$
- Shell Sort: pseudocode is as follows.

Algorithm 1 QuickSelect Algorithm

Input: Array A, indices l and r, integer i
function QUICKSELECT(A, l, r, i)
    if $l = r$ **then**
        return $A[l]$
    end if
    $q \leftarrow$ PARTITION(A, l, r)
    $k \leftarrow q - l + 1$
    if $i = k$ **then**
        return $A[q]$
    else if $i < k$ **then**
        return QUICKSELECT($A, l, q - 1, i$)
    else
        return QUICKSELECT($A, q + 1, r, i - k$)
    end if
end function

Algorithm 2 QuickSort Algorithm

Input: Array A, indices l and r
function QUICKSORT(A, l, r)
    if $l < r$ **then**
        $q \leftarrow$ PARTITION(A, l, r) QUICKSORT($A, l, q - 1$) QUICKSORT($A, q + 1, r$)
    end if
end function

```
Algorithm ShellSort(A: array of items, N: size of array)
Begin
void shellsort(int v[], int n){
    int gap, i, j, temp;

    for (gap = n/2; gap > 0; gap /= 2)
        for (i = gap; i < n; i++)
            for (j=i-gap; j>=0 && v[j]>v[j+gap]; j-=gap){
                temp = v[j];
                v[j] = v[j+gap];
                v[j+gap] = temp;
            }
}
End
```

The idea is to do the insertion sort with a sequence of gaps, starting from something larger. For larger arrays, this can result in efficient algorithm.

It takes $\Theta(n^{1.25})$ in average.

2 Dynamic Programming

- This reduces the problem to smaller problems like D & C but possibly overlapping subproblems. For that reason, we need to memoize.

3 Greedy Algorithm

4 Graph, BFS/DFS, Minimum spanning(Prim's, Kruskal's)

- Three data structures to represent graph: list of edges, adjacency matrix, adjacency lists(could use doubly linkedlist, dictionary, etc).
- We could compare and see the tradeoff between space complexity, finding specific edge, and list of edges of particular node.
- We can devise simple seach algorithms using simple set theory...
- Some more sophisticated search strategies include BFS, DFS, MST, etc.

Algorithm 3 BFS

1. initialize all nodes and put $d[v] = \infty$
2. while $Q \neq \varnothing$
   v = $Dequeue(Q)$
   for each adjacent node of $v \in V$,
$d[v] = d[u] + 1, p[v] = u$, and $Enqueue(Q, v)$

Theorem 2. $\forall v, d[v] =$*length of shortest path from s to v*

Proof. Note that $d[u] = k$ or $k+1$ if $u \in Q$. Further, $d[u] \leqslant k$ if $u \in Done$ for some k. We prove the $\leqslant$ direction since $\geqslant$ is clear. Induct on the length of shortest path from s to v. Let's say this path goes through u. So we have $s - u - v$. By induction hypothesis, we have $d[u] \leqslant$ length of shortest path from s to u.

Note now that $d[v] \leqslant d[u] + 1$ after u is processed. Further, we have $d[u] + 1 \leqslant$ shortest length from s to v by our construction. □

- The key of this proof is to identify the point where u is just processed. This makes the proof rather precise.
- Partition into layers... in this case there will be no "jump" to the figure.
- **DFS** can be used for finding the number of connected components, cyclicity of undirected graph(using parent information), etc
- Can also discuss discovery time and finishing time and their properties (such as being only nested or disjoint).
- We can talk about the structure of the tree: for instance for undirected graphs, there shouldn't be any cross edges.
- We needed to implement stack/queue, and in python, we can just append an element to the array and use the pop method... the operation will take the usual O(1) due to dynamic array in python.
- DFS can be used to find 1) DAG(directed acyclic graphs iff topological sort) and 2) SCC.
- SCC: we do DFS on some s and then do DFS's again in the reverse order of finishing time. The key for the proof is to use the tree structure of DFS.
- If there isn't any back edge on DFS, that means the graph is acyclic

4.1 Dijkstra's algorithm

- The algorithm is based on $d(v) = \min\{d(u) + w(u, v) \mid$ all edges (u,v) $\}$

Miscelleneous

Spectral clustering

Definition 1. *$Ncut(A, B)$ is defined as*

$$Ncut(A, B) = cut(A, B)(\frac{1}{d(A)} + \frac{1}{d(B)})$$

where $d(A) = \sum_{i \in A} d_i$.

Remark 2.
- The intuition is that when A is relatively small, the $\frac{1}{d(A)}$ will be large, hence discouraging the isolating small groups.
- Finding minimum Ncut is equivalent to finding vector v that minimizes

  $$\frac{v^T L v}{v^t D v} \text{such that } v^t D 1 = 0, v_i \in \{a, b\}$$

  where $L = D - W$.
- The thinking goes like this... $M(b, i) = \max\{M(b - w_i, i-1) + v_i, M(b, i-1)\}$ if $b \geqslant w_i$ else $M(b, i) = M(b, i-1)$ where $M(b, i)$ is a maximum value we can get with knapsack of capacity b, uwing a subset of the first i items only.

Algorithms for Linearly Separable Data

There are several ways. First, linear programming.

Algorithm 4 Linear Programming for Classifier

$$\textbf{Objective:} \quad \text{minimize} \quad \mathbf{u} = (0, \ldots, 0) \quad \text{(dummy variable)}$$

$$\textbf{Subject to:} \quad A\mathbf{w} \geqslant \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$A_{i,j} = y_i x_{i,j} \quad \text{(where } j\text{'th element of the vector } x_i\text{)}$$

Remark 3.
- u is a dummy variable here; we essentially only check if the constraint is satisfied.
- This is only applied for when the data is separable.
- We can formula the regression problem with loss function $l(h, (x, y)) = |h(x) = y|$ using linear programming.

Algorithm 5 Batch Perceptron

```
function BatchPerceptron(x_1, y_1, ..., x_m, y_m)
    w(1) ← (0, ..., 0)
    for t ← 1, 2, ... do
        if there exists i such that y_i⟨w(t), x_i⟩ ⩽ 0 then
            w(t + 1) ← w(t) + y_i x_i
        else
            output w(t)
            break
        end if
    end for
end function
```

Remark 4.
- Note that $y_i(w^{(t+1)}, x_i) = y_i(w^{(t)}, x_i) + \|x_i\|^2$
- The algorithm must stop after at most $(RB)^2$ iterations, where $R = \max_i \|x_i\|$ represents a data spread, and $B = \min\{\|w\| : i \in [m], y_i\langle w, x_i\rangle \geqslant 1\}$ represents margin.
- To prove this, it suffices to show that $1 \geqslant \frac{\langle w^*, w^{(T+1)}\rangle}{\|w^*\|\|w^{(T+1)}\|} \geqslant \frac{\sqrt{T}}{RB}$, which we proceed by bounding numerator and denominator separately.
- We can prove that this bound is tight. For some vector $w^* \in \mathbb{R}^d$, the algorithm incurs $m = (BR)^2$ errors (considering $m = d$).
- Moreover, for $d = 3$, an algorithm can be designed to commit exactly (m) errors for any given $m \in \mathbb{N}$, serving as an upper bound concurrently

Logistic Regression

Definition 2. *Fit the logistic function* $\phi_{sig}(x) = \frac{1}{1+exp(-\langle w,x\rangle)}$ *with minimization scheme*

$$w = argmin_{w\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \log(1 + exp(-y_i\langle w, x_i\rangle))$$

Remark 5.
- The explaratory variable is between 0 and 1, making it interpretable as a probability.
- Appropriate for binary classification.
- Logistic loss function is a convex function so it's efficient to minimize.

Variational Inference and EM algorithm

Definition 3. *Kullback-Leibler(KL) divergence between p.d.f.s g and f is given by*

$$d_{KL}(g\|f) = E_g[\log(\frac{g}{f})]$$

This is always nonnegative and it can be shown by Jensen's inequality. Intuitively, we have more 'confidence' on g whenever g is greater than f, whence the logarithm is positive.

Definition 4. *The ELBO (Evidence Lower-Bound) of a p.d.f. g with respect to an unnormalized p.d.f.* $\tilde{f}$ *is given by*

$$ELBO(g) := E_g[\log \frac{\tilde{f}}{g}]$$

Remark 6.
- Simple algebra yields $ELBO(g) \leqslant \log c$, where c is a normalizing constant.
- Let's think in Bayesian framework; our unnormalized function in this case is $\tilde{f}(\theta) := f(y|\theta)f(\theta)$, so $ELBO(g) \leqslant \log f(y)$.
- This justifies the name "evidence lower-bound" and this helps with the choice of modeling (essentially maximizing ELBO).
- We write ELBO(g), but really, what's omitted is that this is with respect to $\tilde{f}(\theta)$.

Remark 7.
- It follows that $ELBO(g) = E_g[\log f(y|\theta)] - d_{KL}(g(\theta)\|f(\theta))$
- In another Bayesian framework, similar equality (will be used in EM algorithm) is

$$\begin{aligned} &ELBO(g,\theta) \\ &= \int \log(\frac{f(y,z|\theta)}{g(z)})g(z)dz \\ &= -d_{KL}(g(z)\|f(z|y,\theta)) + \log(f(y|\theta)) \end{aligned}$$

The first term promotes matching the data, and the second term promotes matching prior beliefs. The reason we work with log domain is that, except for obvious reasons, it helps with numerical stability.
Indeed, remark 2 motivates what's called EM algorithm.

Algorithm 6 EM Algorithm

1: **Input:** Initialization of θ_0
2: **repeat**
3: **E-step:** compute

$$E_{Z\sim f(z|y,\theta_l)}[\log f(y, Z \mid \theta)] = \int \log f(y, z \mid \theta) f(z \mid y, \theta_l) dz$$

4: **M-step:** compute

$$\theta_{l+1} = \arg\max_\theta E_{Z\sim f(z|y,\theta_l)}[\log f(y, Z \mid \theta)]$$

5: **until** some stopping criterion

Theorem 3. *We have* $\log f(y|\theta_l) \leqslant \log f(y|\theta l + 1)$

Proof. Let $g_l(z) = f(z|y, \theta_l)$. We have

$$\begin{aligned}\log f(y|\theta_{l+1}) &= ELBO(g_l, \theta_{l+1}) + d_{KL}(g_l | f(z|y, \theta_{l+1})) \\ &\geqslant ELBO(g_l, \theta_l) + d_{KL}(g_l | f(z|y, \theta_l)) \\ &= \log f(y|\theta_l)\end{aligned}$$

□

- This shows that likelihood function is non-decreasing for each iteration, and since likelihood is always bounded by 1, we have established the convergence of the algorithm.
- GMM algorithm is a specific instance of this algorithm. Here, w_{ik} can be thought of as latent variable, corresponding to E-step, and computing θ and π corresponds to M-step.
- E-step can be computationally expensive and so we usually use Monte-Carlo method to approximate.

Remark 8. • Let us now venture into variational inference, which we use when approximating the intractable distribution. For the choice of possible sets that f admits in $d_{KL}(g\|f)$, we can use **mean field family**, which assumes the independence for coordinate distributions.

Theorem 4. *Let* $g(x) = \prod_{i=1}^d g_i(x_i)$ *with* $g_{-i}(x_{-i})$ *fixed. Then*

$$g_i^*(x_i) \propto \exp(E_{g_{-i}}[\log f(x_i, x_{-i}])$$

maximizes $ELBO(g)$.

Proof. By routine algebra (we need to use independence at some point), one can show that

$$ELBO(g) = E_{g_i}[\log(\exp(E_{g_{-i}}[\log f(x_i, x_{-i})])) - \log g_i(x_i)] + C$$

and notice that the first term can be phrased as -$d_{KL}(g^*\|g)$, whence $g^* = g$ gives the optimization solution. □

Remark 9. • The intuition is to average out the effect of x_{-i} on log expected value in order to incorporate the independence between g_i's.

- This leads to the CAVI algorithm, which approximates the unnormalized target density $\tilde{f}$. After initialization, updating g_i will increase ELBO for each i, so may iterate until ELBO converges.
- The disadvantage is that it may be computationally expensive and accuracy might be not so good.

Algorithm 7 Fast Fourier Transform (FFT) for Circulant Matrix Multiplication

1: **input:** Circulant matrix $A \in \mathbb{R}^{n \times n}$, vector $x \in \mathbb{R}^n$
2: Compute the FFT of the first column of A, denote it as A_{FFT}
3: Compute the FFT of vector x, denote it as x_{FFT}
4: Perform the Hadamard (element-wise) product of A_{FFT} and x_{FFT} to get the result vector y_{FFT}
5: Compute the inverse FFT of y_{FFT} to get the final result vector y
6: **return** y

Use FFT for circulant matrix

- The time complexity is $O(n \log n)$ (versus the usual $O(n^2)$).
- proof:

$$Cx = c_1 * x = IFFT(FFT(c_1) \circ FFT(x))$$

where * is a circular convolution and $\circ$ is a hadamard product.

Fact 1.

$$\mathcal{F}\{x \cdot y\}$$

is a circular convolution of $\mathcal{F}\{x\}$ *and* $\mathcal{F}\{y\}$.

Example 10. Set $x = y = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$. The equality, at least for the first element, yields since

$$3 + 2\exp\left(\frac{-2\pi i}{3}\right) + 2\exp\left(-\frac{-4\pi i}{3}\right) = 9 = 1 + 4 + 4$$

Fact 2. *The inverse transform is given by*

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot \exp(\frac{i2\pi}{N} kn)$$

- This shows how the vector is decomposed with basis in frequency domain.
- This is the analogous transform of Fourier Series (for a regular periodic function) and Fourier transform (for a regular non-periodic function).
- The essence of the theorem is the interchange between multiplication and convolution.

Hashing

- One example would be to add up all ASCII number, divide by the size of the list, and do the modulo (this whole process is a "hash function") by the size of an array (usually a prime number).
- Now, if we end up getting the same number, we can further use linkedlist to store different nodes.